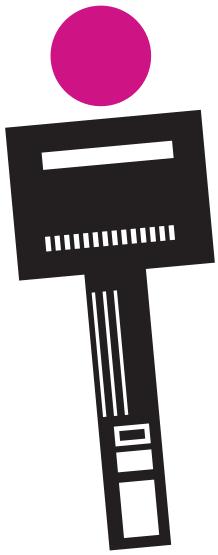

RAM ROBOTICS AND MECHATRONICS

DEVELOPMENT AND TESTING OF A HARDWARE-IN-THE-LOOP SIMULATION INFRASTRUCTURE

F.J.P. (Frank) Hooglander

MSC ASSIGNMENT

Committee:
dr. ir. J.F. Broenink
ing. M.H. Schwirtz
dr. ir. A.Q.L. Keemink

March, 2025

014RaM2025
Robotics and Mechatronics
EEMCS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

UNIVERSITY OF TWENTE. | TECHMED CENTRE UNIVERSITY OF TWENTE. | DIGITAL SOCIETY INSTITUTE

Summary

A Hardware-in-the-Loop (HiL) simulation setup contains a real-time simulator that runs models or systems at the same pace as real life, allowing instant feedback and interaction. In the setup, (part of) a plant is simulated in real-time and used to test an implemented controller. This has safety benefits as it allows thorough testing of the implemented controller without damaging the real system or its surroundings. It also has benefits with regards to costs. In education numerous expensive setups are required when teaching students but HiL simulation enables the replacement of the expensive setups with cheaper Digital Twins. HiL is a helpful tool in Embedded Control System (ECS) development as it bridges the gap between the theoretical simulation and the physical implementation.

The goal of this project is to develop and test a HiL simulation infrastructure. This infrastructure enables HiL simulation for various applications. It contains contributions to the real-time Xenomai framework as well as user guides and recommendations that enable users to perform HiL simulation of their own application. The second goal is to use this infrastructure to develop a Digital Twin of the RELbot which is a mobile robot frequently used at the Robotics and Mechatronics research group. This mobile robot has two motors driving two wheels and it has a castor wheel. The third goal is to measure performance of the Digital Twin with a credibility assessment method that generates a score between 0 % and 100 %.

This project uses a Raspberry Pi 4 with an Icoboard FPGA as the hardware for the Digital Twin. This hardware combination is also used by the ECS. An existing real-time framework developed for the ECS has been used and expanded for use on Digital Twins. This framework facilitates a firm real-time loop on a Raspberry Pi and enables communication to the FPGAs and other components in the setup.

The HiL infrastructure has been developed and tested and a Digital Twin of the RELbot has been developed. Two types of tests have been performed. Firstly, the 20-Sim simulation has been compared with the results of the Digital Twin. This shows how well the 20-Sim simulation on a computer compares to a real-time capable model on a Raspberry Pi. Secondly, the Digital Twin has been compared with the real RELbot to determine if the RELbot model that has been developed is good enough and if the Digital Twin as a whole has good performance.

The results from the first set of tests show that the difference in output between the 20-Sim simulation and the Digital Twin simulation is 7×10^{-3} %. This is a really small error and it is below the 1 % difference requirement. The results from the second set of tests show that the difference in output between the Digital Twin and the RELbot is 1 %. This is also a really small error and meets the requirement of an error below 5%.

The credibility assessment method was used to generate a score for the Digital Twin. The credibility score of the Digital Twin is 97 %.

In conclusion, the goals are achieved and the requirements are met. The HiL simulation infrastructure can be used to develop Digital Twins. The Digital Twin of the RELbot can be used in education. The credibility assessment of the Digital Twin is also completed.

It is recommended to improve the RELbot model by choosing more suitable reference frames and modeling the wheels as separate rigid bodies. It is recommended to expand the Digital Twin framework with a Quality-of-Service check that tracks missed deadlines and informs the user of unreliable results when too many missed deadlines occur. It is also recommended to expand the live visualisation of the RELbot to include the angle of the RELbot. The final recommendation is to test the infrastructure with different plants.

Contents

1 Introduction

1.1 Context

Limited availability and high costs of equipment and physical setups bottleneck the development and testing of controllers in research and education tasks. Such equipment is often task-specific and applicable in a narrow range of applications. There is a need for equipment for education and research facilities that is universally applicable for a wide spectrum of tasks.

Hardware-in-the-Loop (HiL) simulation can be used instead of physical setups to develop and test controllers in order to remove some of the bottlenecks. HiL simulation incorporates crucial hardware that is involved in the actual system and combines this with real-time simulation of (part of) the plant rather than simulating purely in a simulation environment (Isermann, Schaffnit and Sinsel, 1999). Real-time means that the simulation runs at the same pace as real life, allowing instant feedback and interaction. A real-time simulator that emulates a plant is also called a Digital Twin of the plant (Grieves, 2016).

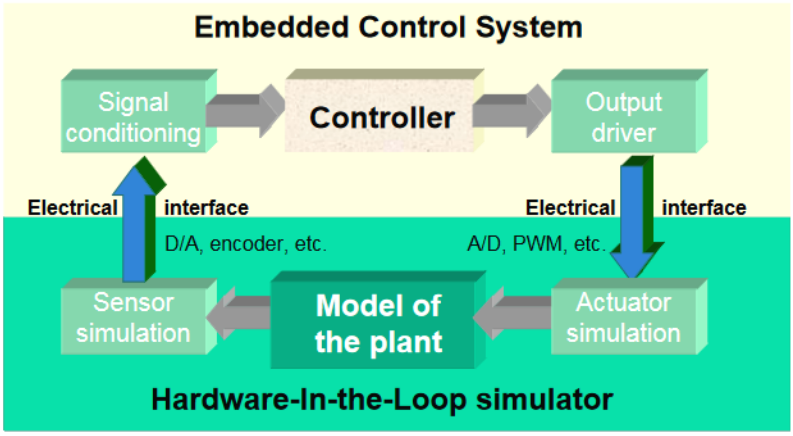

Figure 1.1: HiL simulation setup (Groothuis, 2004)

Figure 1.1 shows a general HiL simulation setup. It shows that the controller and plant are connected with the same electrical signals as the actual physical system. The theoretical controller can be tuned and tested in a simulation environment, but HiL simulation allows testing of the implemented controller on a microcontroller.

At the Robotics and Mechatronics (RAM) research group at the University of Twente a mobile robot called RELbot is frequently used for education and research. A HiL simulation infrastructure can aid in research and the mobile robot is a fitting test case with the added benefit that the Digital Twin can be used in education. Since the mobile robot is available for this project, the research can be taken a step further and the developed Digital Twin can be compared to the real robot.

1.2 Related work

HiL simulation

Groothuis (2004) showed that a HiL simulation setup for a robotic system can yield good results. The setup consisted of a DC motor with a belt and a load. The used hardware is not as powerful as what is available today but good enough for a simpler setup. Yeh and Nugroho (2021) applied HiL simulation to a mobile robot with skid-steering. The controller communicates with a computer that runs a simulation using Unreal Engine. Instead of a computer our research will use a Raspberry Pi to perform HiL simulation. Widiarta, Romdlony, Rosa and

Trilaksono (2021) uses HiL simulation in the development of a controller for a mobile robot focussing mainly on avoiding barriers. Březina and Jabłoński (2018) developed a universal HiL test platform for mechatronic systems using a PLC and tested this with a case study involving a DC motor drive controller.

Real-time framework
The real-time framework establishes a real-time loop which is required for real-time simulation. Delgado, You and Choi (2019) used a Raspberry Pi 3 with Xenomai 3 to create a firm real-time loop and they established communication using Robotic Operating System (ROS). Raoudi (2024) developed a Xenomai framework for firm real-time tasks using a Raspberry Pi 4 with Xenomai 4 and he used ROS2 for communication. The Xenomai 4 framework is used by the RELbot.

I/O interface
The I/O interface corresponds to the sensor simulation and actuator simulation blocks in Figure 1.1. A. I. Pop, N. Pop, Ţarcă, Lung and Sabou (2023) developed a quadrature encoder emulator using a low-cost dual-core microcontroller for HiL simulation of a mobile robot. When generating quadrature signals above 10 kHz the performance was found to be unsatisfactory with both frequency and phase errors. Groothuis (2004) showed that Field-Programmable-Gate-Arrays (FPGAs) can be used to emulate encoder signals with high reliability. The FPGAs are more expensive but have better high-frequency performance and reliability. Hooglander (2023) also did encoder emulation with FPGAs that are compatible with Raspberry Pi 4s, which are also used in the RELbot.

HiL simulation evaluation
Often HiL simulation is evaluated in a qualitative manner where the shape of the output plot of the simulation is compared to the shape of the output plot of the HiL simulation. Dai, Ke, Quan and Cai (2021) developed a credibility assessment method which can be used to evaluate HiL simulation setups in a quantitative manner. The method generates a score of a Digital Twin and these scores can be used to compare different Digital Twins.

1.3 Goals

The main goal is to develop a Hardware-in-the-Loop simulation infrastructure and test this infrastructure by creating a Digital Twin of a mobile robot. The goal is split into three subgoals:

1. Develop and test a Hardware-in-the-Loop simulation infrastructure.
2. Use the infrastructure to create and test a Digital Twin of the RELbot.
3. Evaluate the Digital Twin using a credibility assessment method.

1.4 Approach

An analysis of the problem and the context is done to come up with a set of requirements. Throughout the project a clear distinction is made between the infrastructure and the case study. The infrastructure consists of the parts that are not application specific. The infrastructure is developed and tested first, and afterwards a case study is done. In the case study a Digital Twin of a mobile robot is developed, tested, and evaluated with a credibility assessment method.

1.5 Outline

The rest of the report is structured as follows:

- Chapter 2 discusses the background information on used work that is done by other people.
- Chapter 3 discusses the analysis of the target groups, some aspects of real-time simulation and ends with the requirements.
- Chapter 4 discusses the design of the HiL infrastructure to achieve the goals and meet the requirements for the general HiL simulation infrastructure.
- Chapter 5 discusses the tests of the HiL simulation infrastructure, the results and the interpretation.
- Chapter 6 discusses the design of the casestudy to meet the goals and requirements of the Digital Twin.
- Chapter 7 discusses the tests of the casestudy regarding the Digital Twin performance, the results and interpretation.
- Chapter 8 discusses the conclusion of this project with regards to the goals and requirements and recommendations regarding future research.

2 Background

2.1 RELbot

The RELbot is a mobile robot with two motors with quadrature encoder sensors. The RELbot uses differential drive to steer and has a castor wheel for stability. This robot is developed by the Robotics and Mechatronics research group at the University of Twente. The RELbot is used as a tool in education and as a platform for research and master thesis projects. The RELbot has a Raspberry Pi 4 with an Icoboard FPGA as input/output device.

2.2 FPGA input/output device

An FPGA is a chip containing many programmable look-up tables. The chip can be programmed and synthesised to perform very fast parallel execution of code. The FPGAs use Verilog code. The FPGAs of the Icoboard are used to perform high speed input and output processing of the electrical signals with high precision. For the actuator signals the controller outputs a PWM signal using an FPGA and the Digital Twin reads the PWM signal using an FPGA. For the sensor signals the Digital Twin generates encoder pulses using an FPGA and the controller reads encoder pulses using an FPGA. The module that generates the encoder pulses is called the encoder emulator. The Verilog code for the PWM reader and encoder emulator which has been used and further developed in this project was developed by Hooglander, 2023.

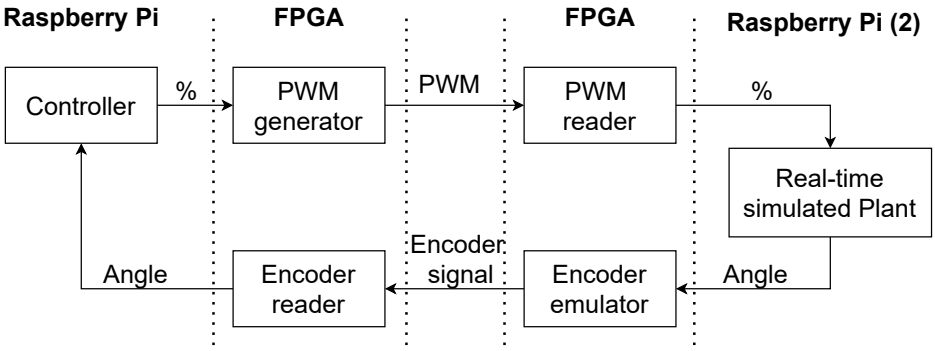

Figure 2.1: FPGA communication

2.2.1 FPGA encoder emulator

The encoder emulator operates on discrete time steps but emulates a real sensor which operates in continuous time. The encoder emulator code calculates how many encoder pulses the encoder position has changed in the previous time-step and sends that amount of pulses in the current time-step. To do this it generates pulses at a frequency of the amount of pulses multiplied by the cycle frequency during a single cycle period.
For example, the update frequency is 1 kHz:

1. At t = 0 ms the encoder position is 0.
2. At t = 1 ms the encoder position is 18.

Between t = 1 ms and t = 2 ms the code generates pulses at a frequency of 1 kHz * 18 = 18 kHz. So the amount of pulses sent in that period is 18.000/s * 0.001 s = 18 pulses.

2.2.2 PWM reader

The PWM reader measures the amount of time per PWM period that the PWM signal is high. The period counter is a 12 bit counter so one period is 4096 clock cycles. The Icoboard has a clock frequency of 100 MHz so the PWM frequency is 100 MHz/4096 = 24.4 kHz. The PWM reader counts how many counts of the 4096 the input signal is high. The duty-cycle is stored as a 11 bit number so the high count is divided by 2 to get the PWM value. If it is high for 2048 counts (and low for 2048 counts) the PWM value is 2048/2 = 1024 which represents a duty-cycle of 50 %.

2.3 ECS framework

The ECS framework is developed by Raoudi (2024). The framework does the following things:

1. The framework runs a firm real-time loop on a Xenomai kernel on a Raspberry Pi.
2. From the firm real-time loop the controller model calculation is called.
3. It enables communication between the Raspberry Pi and the FPGA.
4. It enables communication between the real-time loop and ROS.
5. It has data logging functionality.

2.3.1 Xenomai

Xenomai is a real-time extension for the Linux kernel that provides hard real-time capabilities. It is commonly used in robotics, industrial automation, and other applications where strict timing constraints are required. Xenomai typically runs alongside a Linux kernel using a dual kernel architecture where Xenomai executes the real-time tasks while the non-critical tasks continue running on standard Linux.

2.3.2 ROS

ROS (Robot Operating System) is a flexible framework for writing robot software. Despite its name, it is not an operating system but rather a middleware that provides tools, libraries, and conventions to develop robotic applications. ROS is used on the Linux side of the Raspberry Pi for the robot software.

2.4 Credibility assessment

The credibility assessment method developed by Dai, Ke, Quan and Cai (2021) uses measurement data to generate a credibility score. First a passing score is set which defines the minimal score that is still sufficient with a recommended value of 60 %. Then an error threshold is generated using a confidence interval parameter value with a recommended value of 5 %. The confidence interval represents the acceptable error percentage. A scaling factor ensures the score equals the passing score when the error equals the error threshold. The total credibility score is the weighted average between the time domain score, the performance domain score and the frequency domain score. The weights are chosen by the user.

1. **Time domain**
   The time domain credibility score uses the mean error over time. It uses this value and the error threshold to generate a score between 0 % and 100 %. The confidence interval in the time-domain represents the upper limit for the mean error over time where the error is the difference between the simulation data and the experiment data. If the mean error is 5 % of the total movement, a score of 60 % is given if the recommended parameters are used.
2. **Performance domain**
   The performance domain credibility score compares performance metrics like overshoot, time constant, and settling time. These metrics are used to generate a score

between 0 % and 100 %. A sufficient score is given when the difference between the performance metric in simulation and the performance metric in the experiment is less than 5 %. The performance domain score is the average of the scores generated from the various performance metrics.

3. **Frequency domain**
   The frequency domain credibility score compares phase and magnitude metrics to determine a score between 0 % and 100 %. Once again the difference between the simulation and the experiment must be less than 5 % for a sufficient score. The frequency domain credibility score is the average of the scores calculated from the phase and magnitude data.

2.4.1 Time domain formulas

Since our research only uses the time domain credibility score, the formulas for the time domain credibility are shown while the performance and frequency domain formulas are not. The Root-Mean-Square error between the experimental data y_e and the simulation data y_s is used to find the error of the experimental curve of the time domain e_t.

$e_t = \sqrt{\frac{\sum_1^{n_t}(y_e(t_i)-y_s(t_i))^2}{n_t}}$

The time domain error threshold is found by multiplying the maximum range of the experimental curve with the percentage confidence interval K_p.

$\epsilon_t = K_p * \max_{1\leq i,j\leq n_t} |y_e(t_i) - y_e(t_j)|$

K_e is a normalisation factor that ensures the score is equal to the passing score when the error is equal to the error threshold.

$K_e = \frac{\eta_{pass}}{\sqrt{1-\eta_{pass}^2}}$

The time domain credibility score is then calculated as follows.

$\eta_{time} = \frac{K_e * \epsilon}{\sqrt{(K_e*\epsilon)^2+e^2}}$

When $e = \epsilon$, it results in $\eta_{time} = \eta_{pass}$.

3 Analysis

3.1 Introduction

This chapter discusses the target groups, modeling limitations, the influence of delay of the Digital Twin, the credibility assessment method, and concludes with a set of requirements.

3.2 Target groups

The main target group for the HiL simulation infrastructure is educational institutions. Within the main target group, four sub-groups can be distinguished:

1. Educators
2. Students
3. Researchers
4. Developers

The needs and wants differ per target group so they are analysed to establish the requirements for the design. Some requirements are common for all target groups.

All target groups benefit from high accuracy and reliability of the HiL simulation. The infrastructure should enable users to switch their controller from the Digital Twin to the real setup without making any software adjustments.

3.2.1 Educators

Educators are teachers and teaching assistants who use HiL simulation as part of their course material to alleviate some of the bottlenecks that come with practical education. Think of limited lab space, limited number of physical setups, and the required supervision of students required to ensure safety while working with expensive equipment. The HiL simulation setup can be used to quickly test if the student is ready to move on to the real robot. A clear separation between the students own implementation and the parts provided by the infrastructure makes grading the students work easier. Some courses use the 20-Sim simulation environment in the development of plant models and controllers which means that the usefulness of the HiL simulation infrastructure is improved if it is compatible with 20-Sim.

3.2.2 Students

Students use HiL simulation as part of a course. For some courses a Digital Twin is provided and only used to test the students work but the infrastructure could also be used in different scenarios. Various parts of the HiL setup can be left for the students to implement their own work. The students are not expert programmers so the infrastructure should be easy to use. Monitoring and live visualisation can help the students to gain insight in what is happening which improves the learning process.

3.2.3 Researchers

Researchers use the infrastructure to perform HiL simulations for their own research. Developing the HiL simulation infrastructure to be modular and flexible enables a wider range of use cases, which is important for researchers. The HiL simulation infrastructure should allow different update frequencies to be selected so that applications of various bandwidths can use it. The characterisation of the HiL simulation infrastructure is important so that researchers can weigh the benefits of using it for their research.

3.2.4 Developers

Developers improve and expand the HiL simulation infrastructure so modularity and flexibility of the code is important. Modularity and flexibility make adjustments or expansions to the infrastructure easier. Data logging functionality on the Digital Twin enables the testing of the FPGA communication. For most users, data logging on the ECS is sufficient.

3.3 Real-time simulation

A HiL simulation setup contains a real-time simulator that runs models or systems at the same pace as real life, allowing instant feedback and interaction. A simulation step size of 1 ms means the simulator must run at 1 kHz and for a simulation step size of 10 ms the simulator must run at 100 Hz to make sure one second in simulation takes one second in real life. To keep up with the pace of real-time the calculated states of the next time step must be finished before the deadline at the end of the current time step. Consequently, a proper balance must be found between the computational power of the processor and the computational complexity of the model and integration method.

3.3.1 Implicit vs explicit equations

An equation where the output depends on itself is called an implicit equation and it is solved using iteration until the output converges. An explicit equation does not depend on itself and therefore does not require iteration. Iteration is required when either the integration method or the model is implicit or when both are implicit (Broenink, 2020, Chapter 2.9). Iteration is computationally expensive and not real-time friendly since there is no guarantee that the iteration is finished before the deadline.

3.4 Digital Twin hardware platform constraint

The hardware platform of the Digital Twin is a Raspberry Pi with Icoboard FPGA, which is a project constraint. This combination of hardware is frequently used at the RaM group and it is also used in the RELbot for example.

A Raspberry Pi is limited in computational power which is a drawback of the hardware choice and a limiting factor in the possible implementations of the Digital Twin. Model calculations must be finished before the deadline at the end of the cycle in order to do real-time simulation. A powerful processor can do complex calculations like solving implicit equations using iteration while still finishing before the deadline. The Raspberry Pi might not be able to do this. The limited computation power of the Raspberry Pi requires the use of an explicit model on the Digital Twin.

3.5 Computationally-effective modeling

The following optimalisations lead to an explicit and computationally-effective model.

1. **Model simplification**
   Simplify large-scale or complex systems to reduce computation time and meet deadlines by order reduction, linear approximation of non-linear dynamics, and removal of non-essential dynamics.
2. **Avoid dynamics with high stiffness**
   High stiffness can introduce high frequency oscillations which require very small time steps for stability.
3. **Avoid dynamic causality changes**
   Explicit methods may fail if the causal assignment changes during operation due to, for example, switching power electronics or mechanical collisions.

4. **Ensure preferred causality**
   Explicit models have preferred causality for all energy storage elements. Remove, combine or add elements to ensure preferred causality.
5. **Remove algebraic loops**
   Explicit models have no algebraic loops. Remove, combine or add elements to remove algebraic loops.

Based on bond-graph books like Broenink (2020). A more detailed guide is found in Appendix A.

3.6 HiL Simulation Setup

Figure 3.1 shows the HiL simulation setup. It shows how the Digital Twin is equivalent to the real robot and how the FPGAs are used for signal interfacing. The ECS can be connected to the real robot or to the Digital Twin without any adjustments to the ECS.

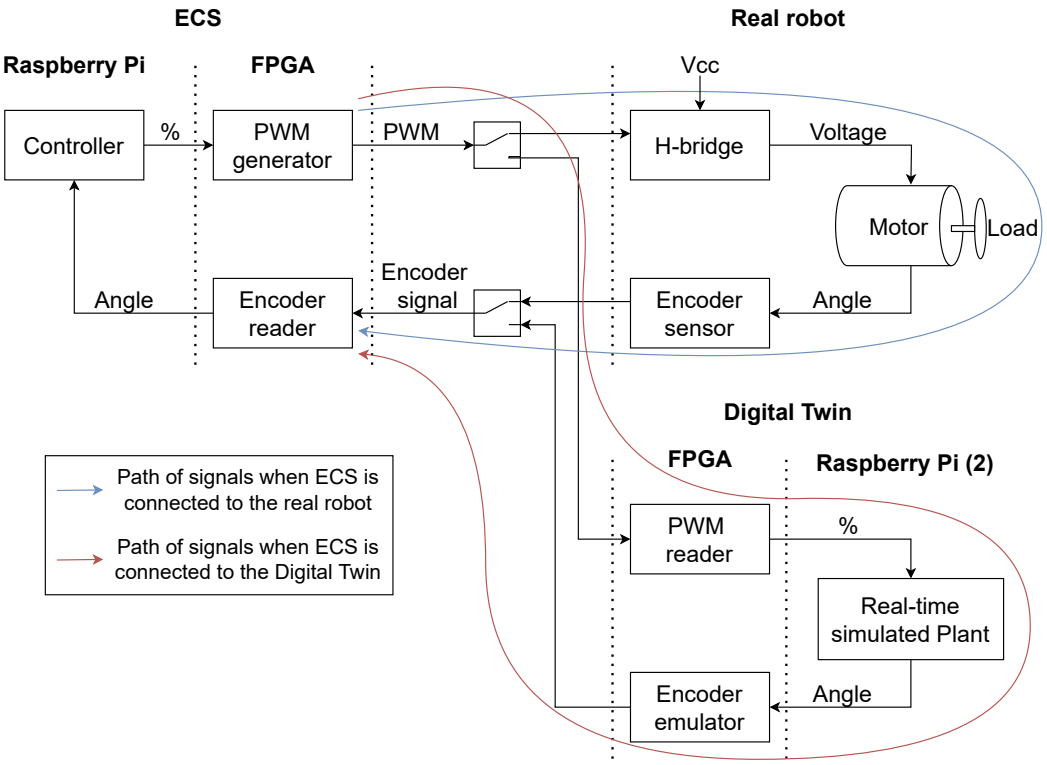

Figure 3.1: HiL Simulation Setup

3.7 Digital Twin delay

The Digital Twin delay refers to the difference in time it takes the Digital Twin to generate outputs based on inputs compared to the real plant. The real plant will almost immediately start generating encoder signals when a voltage is applied and in this section it is assumed to be instantaneous.

A Digital Twin has an unavoidable delay due to its discrete nature. Figure 3.2 shows how the selected update frequency influences the delay of the Digital Twin. The Digital Twin reads and writes the inputs and outputs at the start of each cycle. This is the sampling moment. The input is only sampled once every cycle which results in a delay between zero and one cycle period. Then the calculation of the output takes place which is outputted at the next sampling moment

resulting in a delay of one cycle period. The total delay is therefore between one and two cycle periods.

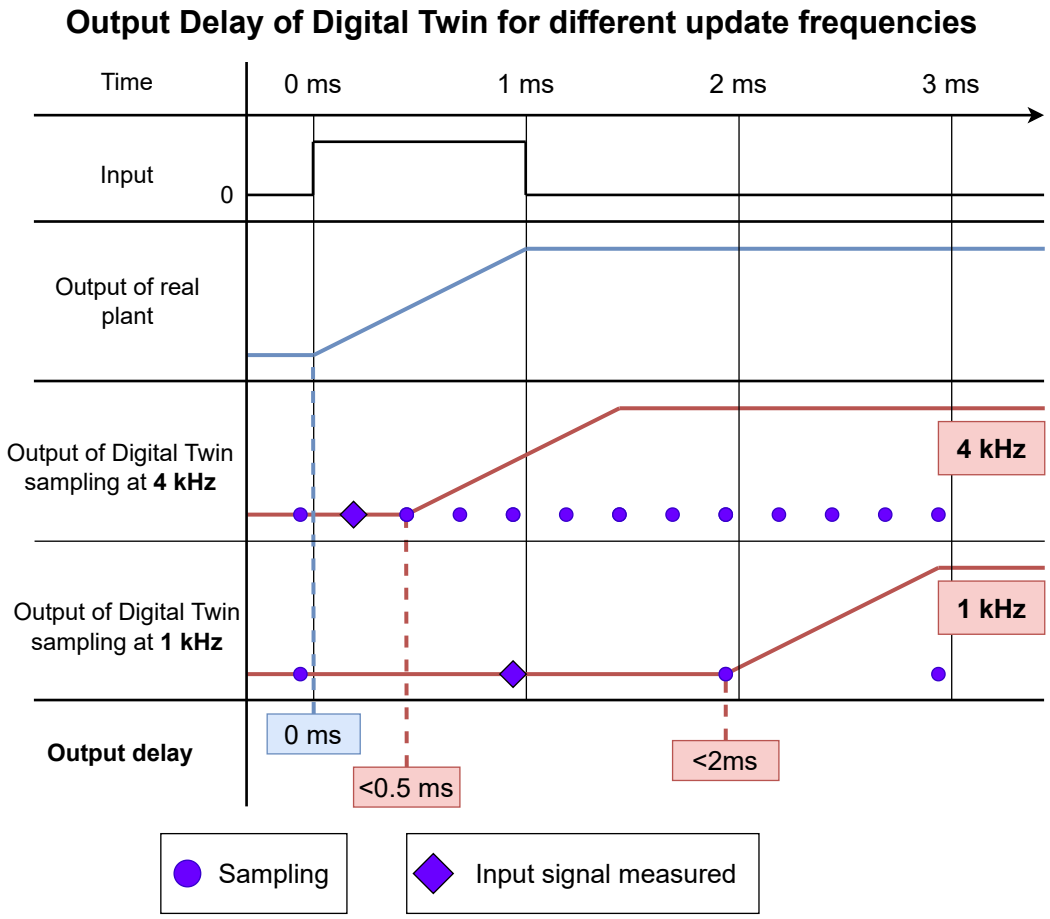

Figure 3.2: Relation between update frequency and delay of the digital twin.

The ratio of the controller sampling frequency and the Digital Twin sampling frequency determines how much of the feedback signal has arrived at the controller one sample moment after sending a signal to the Digital Twin. When the Digital Twin updates at the same frequency 0 % of the feedback has arrived at the next sampling moment. 0-50% at two times sampling frequency, 50-75% at four times sampling frequency, 75-87.5% at eight times sampling frequency.

3.8 Simulation parameter selection

3.8.1 Integration methods

There are three options when it comes to selecting an explicit integration method in 20-Sim: Euler, Runge-Kutta 2 or Runge-Kutta 4. Euler is the simplest and has the lowest accuracy. Runge-Kutta 2 and 4 are more accurate and more complex as they use two and four model-calculation steps respectively. Runge-Kutta 4 is the most accurate and should be considered the default option.

3.8.2 Selecting the simulation time-step for the Digital Twin

Choosing the right simulation time-step requires balancing real-time constraints, stability, and accuracy in the HiL setup. The update frequency and time-step is application specific. Two methods are suggested to find an initial update frequency to start testing:

Method 1: Heuristic approach without performance measurements

1. If measuring performance limits is not feasible, a good rule of thumb is to set the simulator time-step 4 to 10 times faster than the controller's time-step.
2. Validate with HiL testing by measuring the percentage of missed deadlines. If missed deadlines are below 1 %, the time-step is likely appropriate. If above 1 %, the frequency is likely too high and should be adjusted or the model should be simplified.

Method 2: Performance measurements based approach

1. Before running the performance test, it is recommended to set the CPU scaling governor to performance mode. This makes sure the CPU runs at its maximum frequency, finishing the calculation in the shortest time possible. Use the following command to set the performance mode:

   ```
   echo performance | sudo tee \
   /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor
   ```

2. Measure the average calculation time and overhead per cycle of the Digital Twin.
3. To set the CPU scaling governor back to the default schedutil use this command:

   ```
   echo schedutil | sudo tee \
   /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor
   ```

4. Compute the theoretical maximum update frequency as the inverse of this duration. Running at this frequency would cause frequent missed deadlines, as some cycles take longer than average.
5. To set a practical update frequency, start with 75 % of the theoretical maximum. If this results in a frequency much higher than needed (e.g., 100× the controller's frequency), limit it to 4 to 10 times the controller's frequency to avoid wasting computational resources.
6. Validate with HiL testing by measuring the percentage of missed deadlines. This method should result in very few missed deadlines. If missed deadlines are below 1 %, the time-step is likely appropriate. If above 1 %, the frequency is likely too high and should be adjusted or the model should be simplified.

Both methods ensure real-time performance while balancing stability, accuracy, and computational efficiency.

3.9 Credibility assessment

The credibility assessment method by Dai, Ke, Quan and Cai (2021) generates a total credibility score based on the weighted average credibility scores of the performance domain, time domain and frequency domain. They recommend a confidence interval of 5 % which corresponds to an error threshold of 5 % of the total movement in the time domain. The passing score threshold can be set by the user and the recommended value is 60 %.

In the time domain calculation the Root-Mean-Square (RMS) error between the simulation curve and the experimental curve is used to generate a score between 0.00 and 1.00 where 0.00 (0 %) means the RMS error is 150 times higher than the error threshold and 1.00 (100 %) means the average error is 0.075 times smaller than the error threshold. This requires time-stamped simulation and experiment data.

The performance domain calculation uses performance parameters like overshoot percentage, settling time, or steady state error. It compares the performance parameters of the experiment with the simulation to calculate the credibility score. This requires step response data or impulse response data.

The frequency domain credibility compares the magnitude and phase of the simulation and the experiment. This requires frequency response data from the experimental setup which could be done with a frequency sweep. Performance and frequency domain analysis require quite some effort in the data collection so the credibility assessment will be done with time domain data. The user selected weights of the credibility assessment score are therefore: time domain 100 %, performance domain 0 %, and frequency domain 0 %.

3.10 Requirements

Below are the requirements based on the analysis and the goals. The requirements are split into three categories corresponding to the three goals.

Requirements for the HiL simulation infrastructure

Must

1. The ECS software for Digital Twin control must be identical to the ECS software for control of the real plant. Only the hardware connection is adjusted to switch between the two, no software adjustments.
2. The HiL simulation infrastructure must accept code generated by 20-Sim.

Should

1. The HiL simulation infrastructure should have high accuracy. The Root-Mean-Square difference between the HiL simulation output and the 20-Sim simulation output should be less than 1% when provided with an identical input signal.
2. The infrastructure should be applicable for a wide range of applications.
3. The infrastructure should be user friendly.
4. The infrastructure should have a clear separation of built-in and user-created parts.
5. The Digital Twin should have a communication bridge between ROS2 and the Xenomai kernel as this is used in live visualisation.
6. The approach of real-time capable modeling should be documented in a user guide.

Requirements for the case study

Must

1. A real-time capable model of the RELbot must be developed.
2. The Digital Twin must simulate the physical plant in real-time.

Should

1. The Digital Twin should have live visualisation.
2. The case study should have 3D visualisation in the simulation environment.
3. The Digital Twin should have a sufficient credibility score.
4. The Digital Twin should have data logging functionality.

Requirements for the Credibility Assessment

Must

1. The credibility of the Digital Twin must be assessed.

Should

1. Software should be developed to perform credibility assessment.
2. The time-domain credibility assessment method should be used.

Will not

1. The performance-domain credibility assessment method will not be used.
2. The frequency-domain credibility assessment method will not be used.

4 Design of the HiL simulation infrastructure

4.1 Introduction

The HiL simulation infrastructure enables the user to incorporate HiL simulation in their Embedded Control System (ECS) development process. Figure 4.1 shows an expanded and more detailed version of the ECS design approach of Broenink and Ni (2012) and is used to highlight the HiL simulation infrastructure. A user that wants to use HiL simulation as part of their ECS development process follows the steps from 1 to 4 in Figure 4.1.

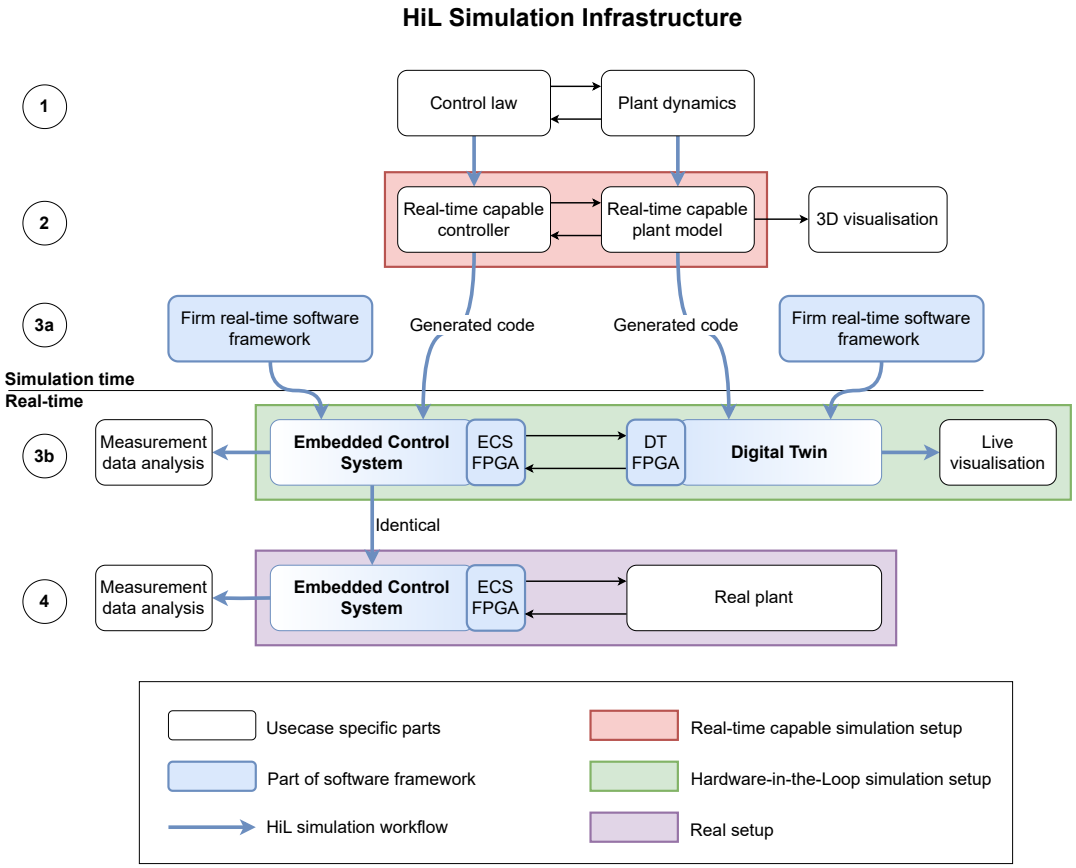

Figure 4.1: HiL simulation infrastructure based on model-driven design approach by Broenink and Ni (2012)

The infrastructure facilitates HiL simulation through a workflow marked with blue arrows and the real-time software framework marked with blue boxes. The ECS and Digital Twin are a combination of the framework and user implementation. Chapter 5 and Chapter 7 are testing chapters. In Chapter 5 the real-time capable simulation setup in the red box is compared to the HiL simulation setup in the green box. In Chapter 7 the HiL simulation setup in the green box is compared to the real setup in the purple box using the RELbot case-study.

This chapter consists of two parts, it first elaborates the expansion of the firm real-time software framework and then it elaborates on the workflow.

4.2 Firm real-time software framework

4.2.1 Combined ECS and Digital Twin framework

The choice is made to have a combined real-time software framework instead of a separate ECS framework and separate Digital Twin framework. The blue boxes in step 3a of Figure 4.1 are the same framework but a few components have a specific version for the ECS and for the Digital Twin. The reason for the combined framework is that from an execution point of view there is no difference between an ECS and a Digital Twin. The main functionality of the ECS software framework is listed below:

1. The framework runs a firm real-time loop on a Xenomai kernel on a Raspberry Pi.
2. From the firm real-time loop the controller model calculation is called.
3. It enables communication between the Raspberry Pi and the FPGAs.
4. It enables communication between the Xenomai and ROS.
5. It has data logging functionality.

The Digital Twin uses a plant model instead of a controller and the contents of the communication with the FPGA are different because the Digital Twin has different inputs and outputs compared to the ECS.

In fact, the inputs and outputs are inversed compared to the ECS. The output of the ECS is an input to the Digital Twin and vise versa. A few classes of the framework that are specific to the ECS I/O have an inverse counterpart for use of the Digital Twin as expansion of the framework.

4.2.2 Implications of adjustable time-step of Digital Twin

Adjustable time-step means that the time-step can be chosen prior to a simulation. During the simulation the time-step does not change. For correct real-time simulation the simulation time-step of the model in the Digital Twin must be equal to the cycle period of the firm real-time loop of the Digital Twin.

The encoder emulator on the FPGA reads the new angle once per cycle period to calculate the angle difference. It uses a counter that overflows once per period to do this. To function properly this counter needs to know the time-step size which means that the FPGA needs to know the time-step size. A more detailed explanation of the encoder emulator is provided in the background in Section 2.2.1. Getting the information of the time-step size to the FPGA is not straightforward.

Possible solutions

The problem of communicating the time-step size to the FPGA has the following possible solutions:

1. **Automatic time-step size communication**
   The first option is automatic time-step size communication between the Raspberry Pi and the encoder emulator on the FPGA. The advantage is that it is really user friendly since the user does not have to do any additional manual actions and it enables a wide range of possible applications. It is also quite robust to user error. The disadvantage is that automatic time-step size communication requires major changes to the Verilog FPGA code as the encoder emulator has to be adjusted and the SPI communication has to be adjusted.
2. **Different frequency presets**
   The second option is to synthesize multiple versions of the FPGA code with different preset update frequencies. This way the user can choose one of the available update frequencies and use the corresponding FGPA code version. The advantage is that it is not as hard to implement as it does not require changing the SPI communication between

the Raspberry Pi and the FPGA and it does not require too many adjustments to the encoder emulator code while still allowing different update frequencies. The disadvantage is that this is prone to user error and it is not as flexible as there is a limited amount of frequencies to choose.

3. **No adjustable time-step**
   The third option is no adjustable simulation time-step. The advantage is that this requires no extra effort in development. The disadvantage is that it limits the range of possible applications.

Automatic time-step size communication

Option 1: Automatic time-step size communication is chosen because of the user friendliness and because it makes the infrastructure useful for a wider range of applications.

The step-size is communicated to the FPGA during the initialisation of the FPGA. The initialisation happens once and it happens before the simulation starts which is exactly when the step-size should be communicated. This initialisation is done with an INIT message. The empty bytes of the INIT message are used to communicate the step-size so a new message type is not needed. The step-size is put in the initialisation message automatically, so the user does not have to do anything.

4.2.3 FPGA code version check

The HiL simulation infrastructure contains two versions of FPGA code, one for the ECS FPGA called *icoboard* and one for the Digital Twin FPGA called *icoboard_inverse*. Figure 4.1 shows that the ECS FPGA is connected to the ECS and the DT FPGA is connected to the Digital Twin. These two versions can be accidentally mixed up by the user while setting up the HiL simulation setup. To improve ease of use, common user errors should be prevented. This is why the design decision is made to add a check during initialisation of the FPGAs that prevents the mixing of the two FPGA code versions.

When the Raspberry Pi sends the SPI INIT message the FPGA code responds with a unique value. The value of the ECS FPGA code is different from the value of the Digital Twin FPGA code. The ECS and Digital Twin throw an error if the wrong value is returned during initialisation. Figure 4.2 shows the error message when the Digital Twin FPGA is connected to the ECS.

```
./build/main/main: Failed to initialise FPGA, use icoboard instead of icoboard_inverse
pi@rel-rpi:~/workspace$
```

Figure 4.2: Wrong FPGA version error message

4.2.4 Improving the Encoder Emulator of the Digital Twin

The FPGA I/O device of the Digital Twin contains a module that reads PWM signals and a module that emulates encoder signals. Although the encoder emulator developed by Hooglander (2023) was quite reliable at 1 kHz with an error below 1 in 50.000 messages, reliability dropped at higher frequencies. The part of the code that determines output frequency of the encoder based on the angle input has been improved. The output frequency calculation consists of three steps:

1. **Angle difference calculation**
   At the start of each time step, sample the new position and calculate the absolute difference between the new and old angle, and the direction.
2. **Overflow correction**
   Since the relative encoder position is used it can happen that -200 steps is calculated as

+16183 steps. This should be detected and corrected. If overflow is detected the direction should be corrected as well.

3. **Scaling with the update frequency**
   The angle difference should be scaled with the update frequency. If 200 steps must happen in 1 ms (1000 Hz) the output should generate pulses at 200*1000 = 200.000 Hz.

FPGA code has blocking (=) and non-blocking (<=) statements. Blocking statements are executed sequentially while non-blocking statements are executed in parallel which is preferred. Hooglander (2023) implemented this calculation with blocking statements which is easier as it consists of three steps that have to be executed sequentially but it is not as reliable. The code has been changed to only use non-blocking statements.

Combined angle difference calculation and overflow correction

The angle difference calculation and overflow detection are calculated at the same time. There are four possible angle difference situations since the difference can be positive or negative and it can have overflow or not.

Table 4.1: Angle difference, overflow and direction overview.

Situation	Difference (new-old)	Overflow	Direction
A	Positive	No	Positive
B	Negative	No	Negative
C	Positive	Yes	Negative
D	Negative	Yes	Positive

This is implemented in the following way in pseudocode:

```
if (A) begin
    clk_encoder_pulses <= angle_difference
    direction <= positive;
end else if (B) begin
    clk_encoder_pulses <= - angle_difference;
    direction <= negative;
end else if (C) begin
    clk_encoder_pulses <= full_encoder_resolution -
   angle_difference;
    direction <= negative;
end else if (D) begin
    clk_encoder_pulses <= full_encoder_resolution +
   angle_difference;
    direction <= positive;
end
```

Scaling the clock divider value

In FPGA implementations, the use of multiplication and division operations should be minimized or avoided whenever possible due to their high resource consumption and latency. The multiplication required to scale the angle difference is removed and instead, the frequency generator uses a clock divider which scales the output to the correct frequency. The clock divider value is provided by the Raspberry Pi during initialisation.

This part of the code responsible for calculating the encoder emulator output frequency was executed on the rising edge of a register which led to race conditions causing reliability issues.

This is adjusted to execute on the rising edge of the clock and then detect if the register is high, removing the final reliability issues. The error of 1 in 50.000 messages is removed and in 720.000 messages, no error was found. The encoder emulator is tested in the next chapter.

4.3 Workflow

4.3.1 Real-time modeling user guide

The workflow consists of steps 1 to 4 as shown in Figure 4.1. This section is about the arrows between these steps which help the user to progress through the workflow. The workflow starts at step 1 of Figure 4.1 with a normal plant and controller model. In order to do HiL simulation using this framework explicit models are needed as shown in step 2. To help users to make their models real-time capable and explicit, a user guide is provided which is shown in Appendix A. The user guide focuses on three aspects.

1. **Model complexity reduction**
   Explains some methods to reduce the complexity of the model and thereby the required computation time of the model.
2. **Model stability**
   Explains the possible problems of high stiffness dynamics when it comes to stability and fixed time-step simulation.
3. **Making the models explicit**
   Explains the implicit relations that can be encountered and how to make them explicit.

4.3.2 Implementation user guide

Now that the user has explicit models at step 2 of the workflow, a second user guide is made about the steps the user should take to do HiL simulation and eventually implement the ECS on the real setup. The user guide is shown in Appendix B. The user implementation and the live visualisation are further elaborated in the next sections. The guide contains the following information:

1. **Code generation**
   Explains how to generate code from the explicit model.
2. **Generated code implementation**
   Explains how to implement the generated code in the framework.
3. **Simulation parameters**
   Explains how to adjust the simulation time-step, the simulation duration, and the integration method.
4. **User implementation**
   Explains what the user has to implement for the parts that are specific to the use case. This is pre- and post-processing of the model inputs and outputs, executing the calculation, and optionally logging.
5. **Output analysis**
   Explains how the logged data from the ECS can be processed and plotted using 20-Sim. Explains how the credibility assessment script is used. The credibility assessment script is shown in Appendix C.
6. **Live visualisation**
   Explains how to adjust the model for live visualisation, how to use ROS topics to communicate the data, and how to do live visualisation.
7. **From HiL simulation to the real setup**
   Explains the steps the user should take to connect the ECS to the real plant.

User implementation

There are four main steps that the user has to do to implement the generated code with the software framework on the Raspberry Pi as shown in Figure 4.3.

1. Preprocessing of the measured input data of the FPGA so it can be used by the model calculation.
2. Calculating the new outputs of the model.
3. Postprocessing of the model calculation output values so they can be sent to the FPGA in the next cycle.
4. Logging the input and output values. This is optional as logging on the ECS side is sufficient in most cases.

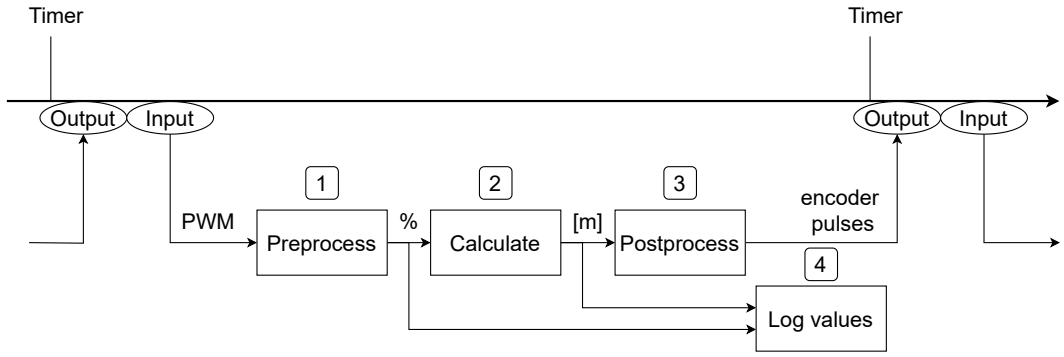

Figure 4.3: Overview of user implementation of code

Live visualisation

A live, visual representation of the simulation is easier to understand than reading changing simulation variables in a terminal. The HiL simulation infrastructure enables implementation of live visualisation. Live visualisation should be kept simple to minimise delay and to achieve acceptable framerates. The centre-of-mass position of a mobile robot or the position of an end-effector are good options. These position values should be outputs of the simulation model which are then sent to a ROS topic. The variable names and datatypes in the ROS message are use case specific and are set by the user. The live visualisation is done with a ROS visualisation package on a computer which is connected to the same network as the Digital Twin.

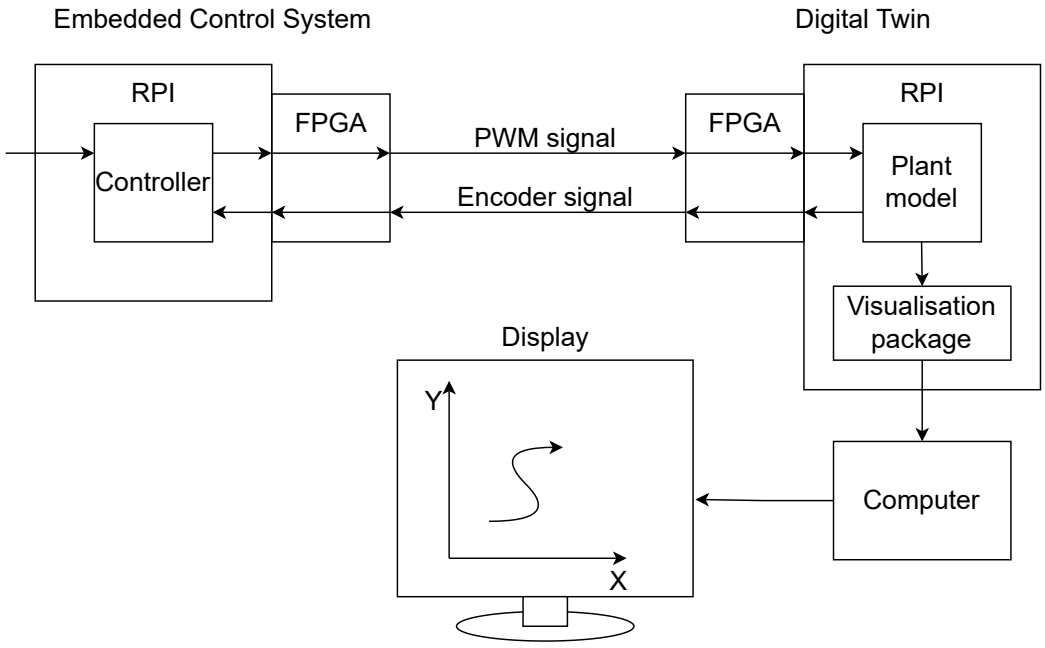

Figure 4.4: Live visualisation setup

5 HiL Simulation Infrastructure Testing

5.1 Introduction

This chapter contains the tests of the infrastructure and the results. The tests and results in this chapter focus on the performance of the Digital Twin platform rather than a specific implementation of a Digital Twin. By comparing the Digital Twin with the 20-Sim environment, modeling errors do not influence the outcome as both platforms use the same model. The errors are a result of the limitations of the Digital Twin platform.

Tests in this chapter compare the Hardware-in-the-Loop simulation setup with the normal simulation setup. The HiL simulations are green as they are part of the green box in Figure 4.1 and the normal simulations from 20-Sim are red. Note that comparative plots showing the difference between the 20-Sim and HiL simulations are also in red.

5.2 Digital Twin vs 20-Sim open-loop test

5.2.1 Goal and setup

The goal of this test is to validate the Digital Twin platform by comparing the open-loop response to an input profile with the response of the 20-Sim model. The input is designed to excite all aspects of the model and it includes sudden input changes as well as gradual input changes as shown in Figure 5.1. The input profile below is about 36 s but this input profile is repeated to perform a test of 10 min.

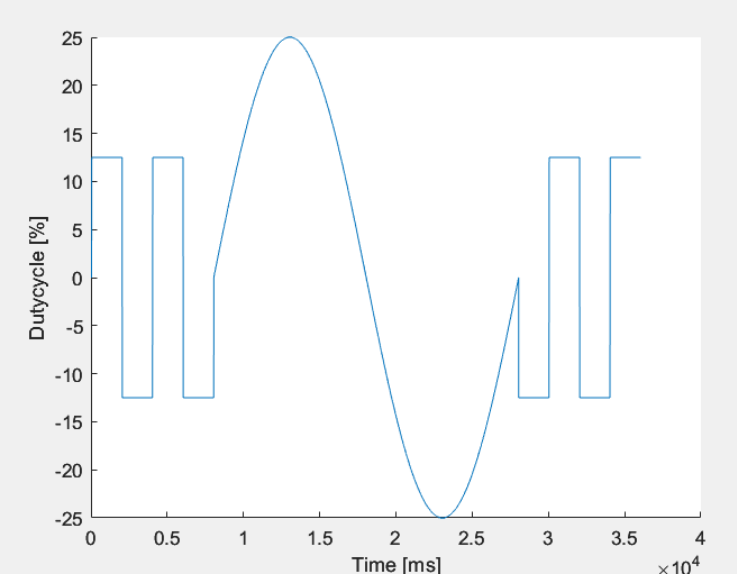

Figure 5.1: Duty-cycle input profile for open-loop comparison test of 20-Sim and Digital Twin.

Figure 5.2 shows the test setup. The ECS reads the input profile into memory before the start of the simulation and during the test it sends the PWM from the input profile to the Digital Twin. The Digital Twin simulates the plant in real-time and returns encoder signals corresponding to the calculated model output. The ECS reads the encoder values and logs both the duty-cycle and position value at the end of each cycle. This data is used as a source file in 20-Sim where the logged duty-cycle is connected to the plant model and the output of the plant model is compared to the outputs in the log file. The test is done with a model of the RELbot and the output that is compared is the position of the right wheel. A model of a different plant could

have been used too as these tests are largely model independent. The test setup in 20-Sim is shown in Appendix F.

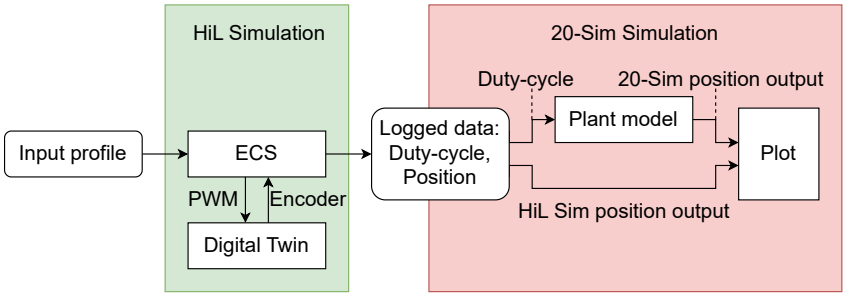

Figure 5.2: Open-loop Digital Twin vs 20-Sim comparison test setup.

5.2.2 Results

Figure 5.3 shows the wheel position output from the Digital Twin in green and the wheel position output from the 20-Sim model in red in the top plot. These curves are very similar so only the green line is visible. The pattern of the position is as expected given the used input profile. The bottom plot shows the difference between the two curves by plotting the Digital Twin values subtracted from the 20-Sim values. The absolute position difference is 2 mm or less.

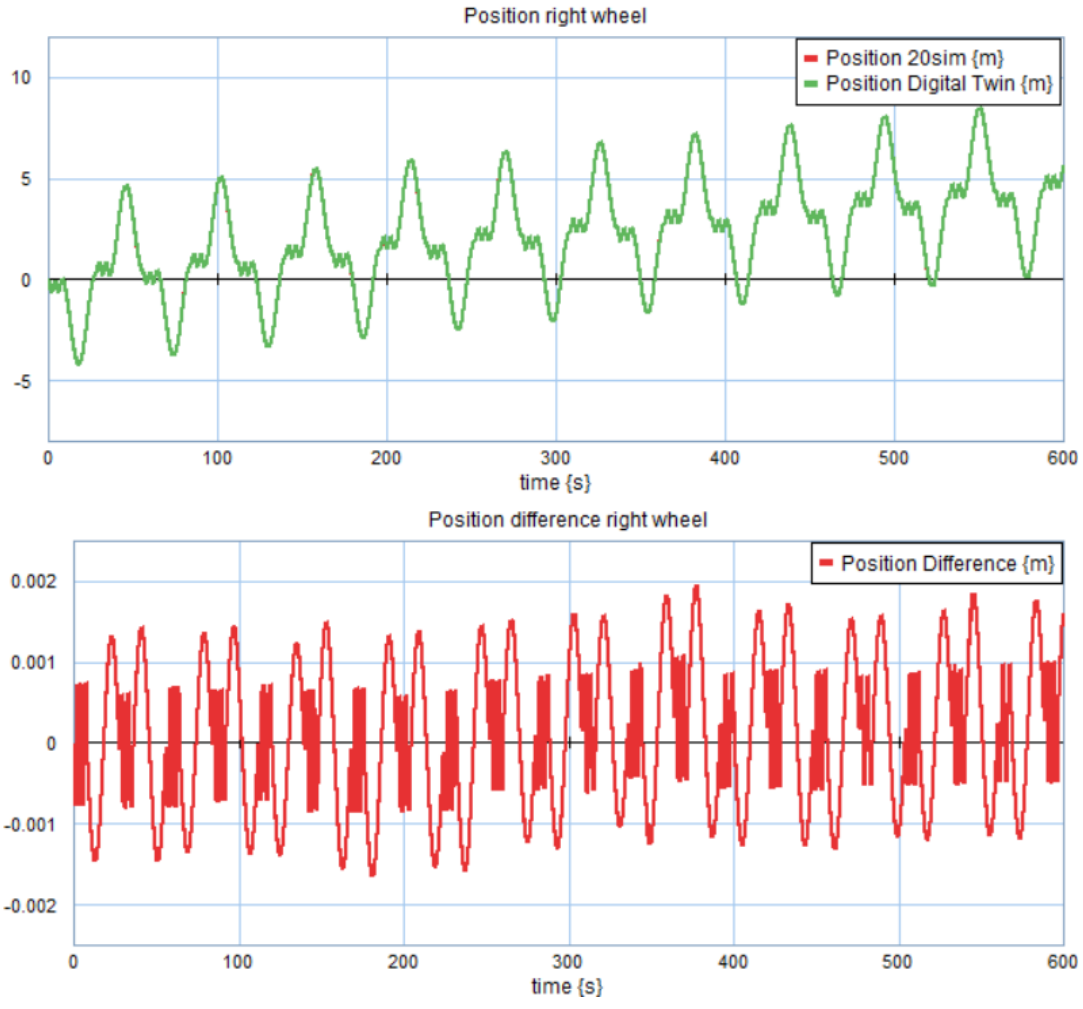

Figure 5.3: Open-loop 20-Sim and Digital Twin comparison

5.2.3 Interpretation

The average difference between the Digital Twin simulation output and the 20-Sim output is 9.22×10^{-4} m or 7.29×10^{-3} %. This means the requirement of an average difference below 1 % is achieved. There is a pattern in the error with a period matching the input profile which could be a result of the delay introduced by the Digital Twin.

5.3 Digital Twin vs 20-Sim open-loop test with post-test delay compensation

5.3.1 Goal and setup

The goal of this test is to find how much of the error of the Digital Twin is a result of the delay. The setup is the same as in the previous test but the delay of the Digital Twin is compensated using a time-delay block in 20-Sim. The test setup in 20-Sim is shown in Appendix F.

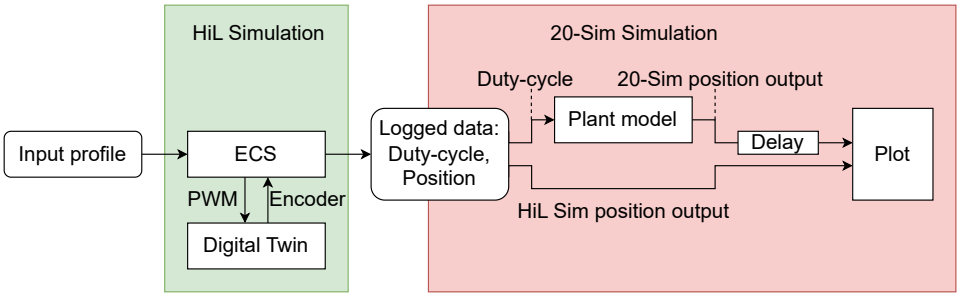

Figure 5.4: Open-loop Digital Twin vs 20-Sim comparison test setup with delay compensation.

5.3.2 Results

Figure 5.5 shows the wheel position output from the Digital Twin in green and the wheel position output from the 20-Sim model in red in the top plot. The curves are very similar so only the green line is visible. It shows the difference between the curves in the bottom plot. The plots show that the absolute value of the position difference between the Digital Twin output and the 20-Sim output is 0.3 mm or less during this test.

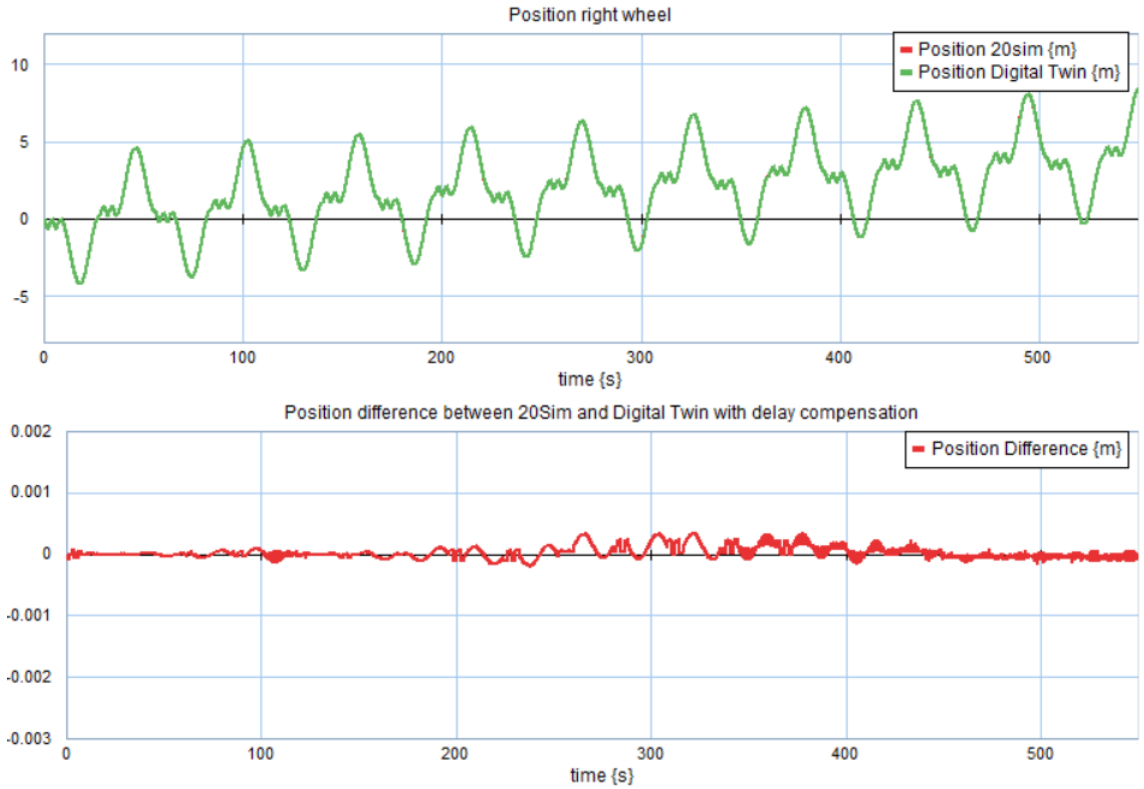

Figure 5.5: Open-loop 20-Sim and Digital Twin comparison without delay.

5.3.3 Interpretation

The 2 mm error becomes 0.3 mm when the 20-Sim data is shifted 2 ms in time resulting in an average error of 9.71×10^{-5} m or 7.68×10^{-4} %. This is ten times smaller than the average error with delay. This shows that about 90 % of the error is a result of the delay. Due to the small average error between the model in 20-Sim and the model on the Digital Twin it can be concluded that the HiL simulation platform works well. This includes:

1. The code generation tool.
2. The real-time framework of the ECS and Digital Twin.
3. The Raspbery Pi to FPGA communication.
4. The PWM generator and PWM reader modules.
5. The Encoder emulator and encoder reader modules.

5.4 Digital Twin update frequency tests

5.4.1 Goal and setup

The goal of this test is to compare the performance of different Digital Twin update frequencies. In Section 3.7 the relation between update frequency and delay of the Digital Twin is analysed and the previous tests of Section 5.2 and Section 5.3 showed that delay increases the output error. The analysis also showed that high update frequencies can lead to missed deadlines. This test uses the same setup as the test in Section 5.2 and performs the same test at four different update frequencies. The errors are plotted and the missed deadlines measurements are shown in Table 5.1.

5.4.2 Results

The four plots show the position difference at 1 kHz, 4 kHz, 8 kHz, and 16 kHz. Table 5.1 shows the total amount of cycles of the firm real-time loop on the Digital Twin, the amount of missed cycles and the percentage of missed cycles for each of the four update frequencies.

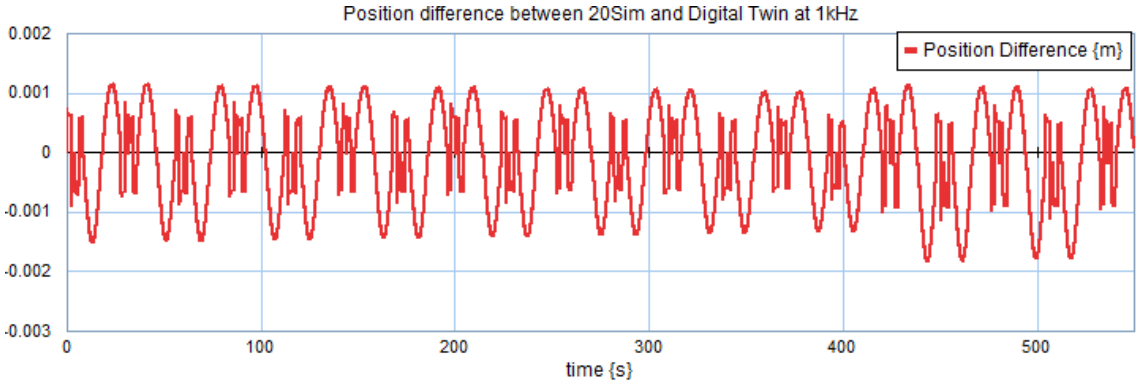

Figure 5.6: Position difference with Digital Twin at 1kHz.

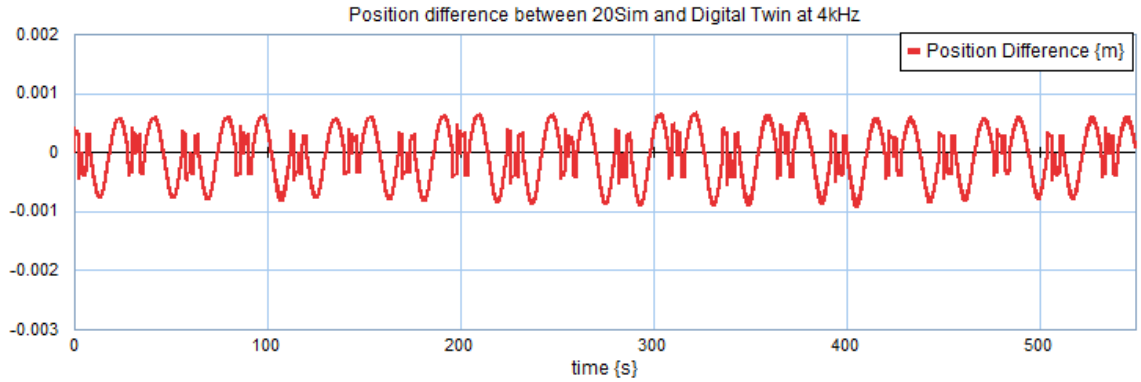

Figure 5.7: Position difference with Digital Twin at 4kHz.

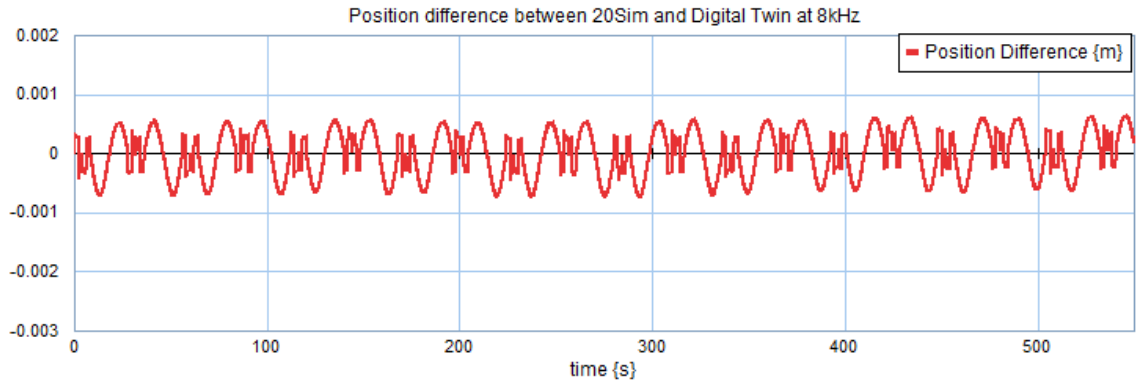

Figure 5.8: Position difference with Digital Twin at 8kHz.

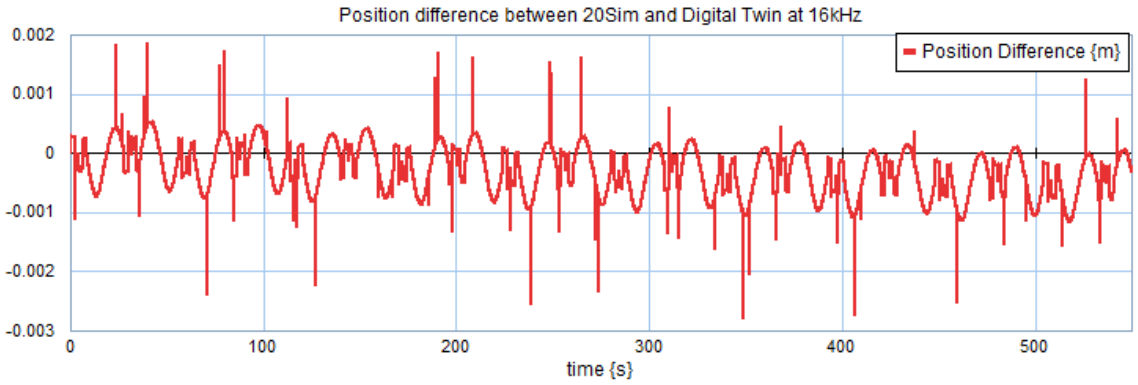

Figure 5.9: Position difference with Digital Twin at 16kHz.

Table 5.1: Percentage of missed deadlines for various Digital Twin update frequencies.

Frequency (kHz)	Total cycles	Missed cycles	Missed percentage (%)
1	600.000	1	1.66×10^{-4}
4	2.400.000	9	3.75×10^{-4}
8	4.800.000	325.762	6.78
16	9.600.000	1.630.674	17.0

5.4.3 Interpretation

The plots show a clear improvement in error between 1 kHz and 4 kHz and a slight improvement between 4 kHz and 8 kHz. Figure 5.9 shows that 16 kHz is too high resulting in unpredictable behaviour. Table 5.1 shows that 1 kHz and 4 kHz have almost no missed deadlines while 8 kHz has 6.78 % and 16 kHz has 17.0 % missed deadlines.

With 6.78 % missed deadlines for the 8 kHz test a noticeable error is expected in plot Figure 5.8, but there is no noticeable error in this particular test. This is why it is important to also look at missed deadlines when selecting an update frequency.

Figure 5.8 suggests that 8 kHz is the optimal frequency but with 6.78 % missed deadlines the reliability is too low. Using the plots and the table it is concluded that 4 kHz is the preferred update frequency as it has a missed deadline percentage below 1 % and a smaller error than the 1 kHz update frequency.

5.5 FPGA encoder emulator test

5.5.1 Goal and setup

The goal is to test the performance of the FPGA encoder emulator module. Encoder values are sent from the Digital Twin to the ECS and the values are logged on both sides. The encoder values of the ECS are subtracted from the encoder values of the Digital Twin and the resulting data is plotted. Any error in the generation of the encoder pulses or the reading of the encoder pulses result in a non-zero data point. Since logging is done on two devices, the data has to be synchronised after the test using recognisable features in the dataset as explained by Hooglander (2023, Section 5.3.1).

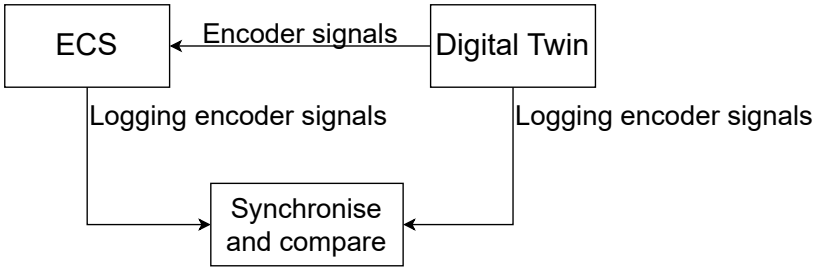

Figure 5.10: Encoder test setup

5.5.2 Results

Figure 5.11 shows the difference between the sent and the received encoder pulses.

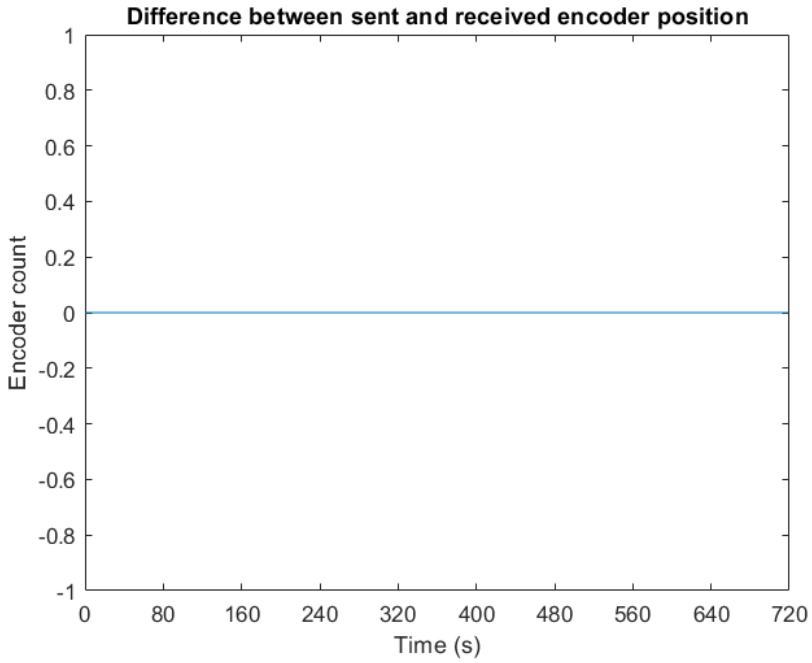

Figure 5.11: Encoder signal communication test

Table 5.2: Encoder emulator and encoder reader reliability test.

Number of messages sent	Number of messages received	Number of messages missed
720.000	720.000	0

5.5.3 Interpretation

Figure 5.11 and Table 5.2 show that no error exists between the encoder emulator and encoder reader during this test since the error is 0 throughout the whole test. Other tests at different frequencies have the same result and are therefore not shown here. This result is possible due to the high reliability and fast parallel processing of FPGAs when proper Verilog code without race conditions is used.

5.6 Credibility assessment of HiL simulation hardware platform

5.6.1 Goal and setup

The goal is to find the credibility score of the HiL simulation infrastructure. This is done by comparing a Digital Twin simulation to the plant model in a simulation environment. This way model inaccuracies will not affect the credibility score as the same model is used on both systems. Delays, calculation inaccuracies and other imperfections of the Digital Twin are the only contributors to the output difference between the Digital Twin and the 20-Sim simulation.

The test compares the open-loop output data of the Digital Twin with an update frequency of 4 kHz with the open-loop output data of 20-Sim given an input profile. The passing score is set at the recommended value of 60 %. The confidence interval is set at the recommended 5 %. The time-domain credibility assessment method is used on the position data of test 5.2 shown in Figure 5.3.

5.6.2 Result

Parameters:

$K_p = 0.05$, $n_{pass} = 0.6$

$K_e = \frac{n_{pass}}{\sqrt{1-n_{pass}^2}} = 0.75$

Measurements:

Error threshold: $\epsilon_t = 0.1854\ m$

Difference between 20-Sim and HiL curve: $e_t = 9.101 \times 10^{-4}\ m$

$\eta_{time} = \frac{K_e * \epsilon_t}{\sqrt{(K_e * \epsilon_t)^2 + e_t^2}}$

$\eta_{time} = \frac{0.75 * 0.1854}{\sqrt{(0.75 * 0.1854)^2 + (9.101 \times 10^{-4})^2}} = 0.999978 = 100\ \%$

5.6.3 Interpretation

The credibility assessment score is calculated to be 100 % which is the highest possible score. This is expected since the average error of 9.101×10^{-4} m is so small compared to the 0.1854 m error threshold. This result is possible because the test excludes model imperfections. The result shows that the Digital Twin hardware platform will not be a limiting factor for the credibility score of HiL simulation setups. This means that the Raspberry Pi with Icoboard FPGA is a suitable hardware platform for a Digital Twin when used with an explicit model.

6 Case-study Design

6.1 Introduction

This chapter describes the design of the case-study with the RELbot. The RELbot shown in Figure 6.1 is a mobile robot developed by the Robotics and Mechatronics (RaM) research group at the University of Twente. This robot is used for educational purposes and research.

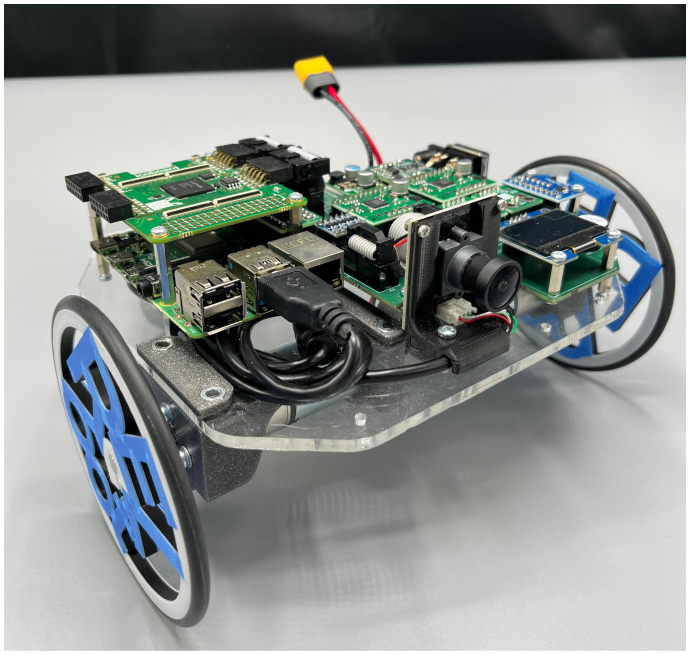

Figure 6.1: Image of the RELbot

6.2 RELbot model

6.2.1 Top level model overview

Figure 6.2 shows a top level overview of the RELbot model in the simulation environment. The Controller receives wheel position feedback and generates steering values. The Plant receives steering values and has the wheel positions as outputs. The full model and all submodels are shown in Appendix D.

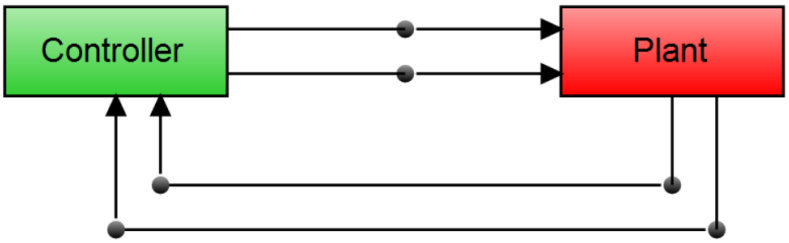

Figure 6.2: The RELbot controller and plant model in 20-Sim

6.2.2 Model reference frame

The reference frame of the model is fixed to the body of the RELbot as shown in Figure 6.3. Positive X is forward and positive Y is to the left of the RELbot. The RELbot can only move around in the X and Y axes. The wheels are modeled in the same reference frame which means they rotate along the Y axis. Positive rotation of the wheels means the RELbot moves forwards in the positive X direction. The origin of the reference frame is in the centre-of-mass of the RELbot. Movement in the Z-axis is not considered but the RELbot can rotate around Z.

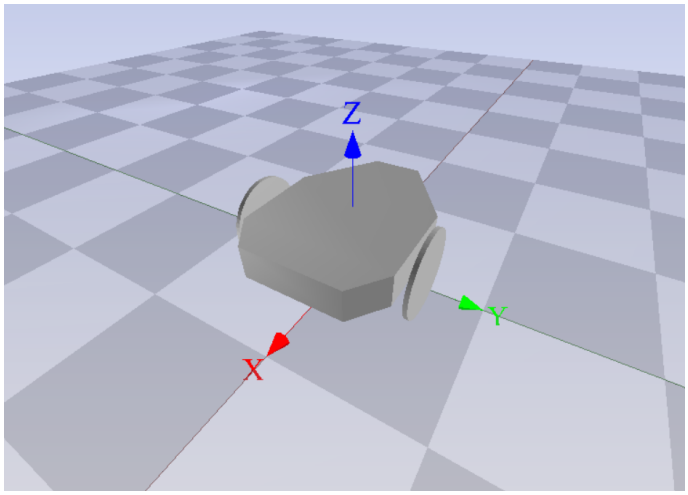

Figure 6.3: RELbot reference frame

6.2.3 Implicit RELbot plant model

In a previous project Wielink (2024) developed a bond-graph model of the RELbot in 20-Sim which is the starting point for the case-study model. This implicit model must be converted to an explicit model so that it can be used for real-time simulation by the Digital Twin.

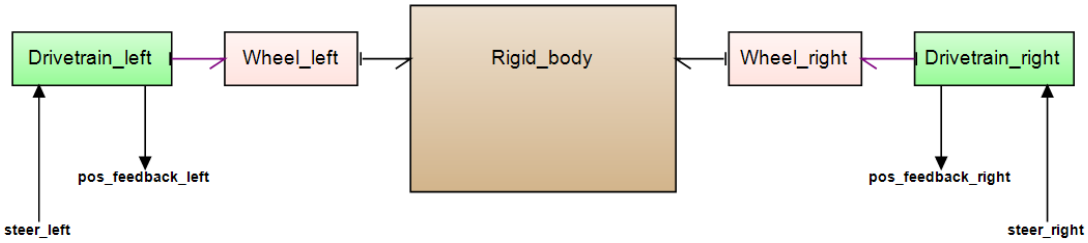

Figure 6.4: The RELbot plant model subsystem overview

The drivetrain models receive steering values from a controller. The drivetrain models calculate how fast the wheels rotate and the wheel models convert the rotation to translation. The rigid body determines how the body rotates and moves based on the inputs from the wheels.

6.2.4 Model input/output datatype selection

The model inputs and outputs could be chosen to match the types used in the FPGA input/output devices to make implementation easier. However, the choice is made to use SI units adhering to standard modeling principles. This means the inputs and outputs require some pro-

cessing between the model and the FPGA interface as mentioned in Section 4.3.2. This is explained in Section 6.5.1.

The input for the model can be a voltage or a duty-cycle value. The choice is made to use a duty-cycle value between -1 and 1 as input to the model. This is because the output of the controller is not a voltage. The duty-cycle is turned into a voltage by the motor drivers that are part of the drivetrain. The output of the model is chosen to be the wheel position in meters, which is the distance the center of the wheels have moved in the X direction with respect to the ground. This is chosen because it is an SI unit and it enables easy visual validation with the real robot.

6.3 Conversion to a Real-Time capable model

The user guide to real-time capable modeling in Appendix A is used to convert the implicit RELbot model to a real-time capable and explicit model. There are causality issues in the drivetrain and wheel model and causality issues in the rigid_body. Solving these issues resulted in an algebraic loop which was subsequently solved.

6.3.1 Drivetrain

Step 1 of the user guide is to simplify the model by removing non-essential dynamics. A motor inductance often has negligible influence on the dynamic of a system so a test is performed comparing the model with the inductance vs without the inductance. This showed that the difference between the wheel position with the inductance and without the inductance is negligible so the inductance is removed to simplify the model. This means the R_actuator now has flow-out causality instead of effort-out. Removing this inductance does not solve any causality issues.

Step 4 of the user guide is to ensure preferred causality which is applied to the various I-elements in the drivetrain and wheel model shown in Figure 6.5. The elements can be combined by pulling it through the transformer (TF) elements. The rotor inertia, the wheel inertia and the wheel mass are combined in an equivalent inertia element to ensure preferred causality.

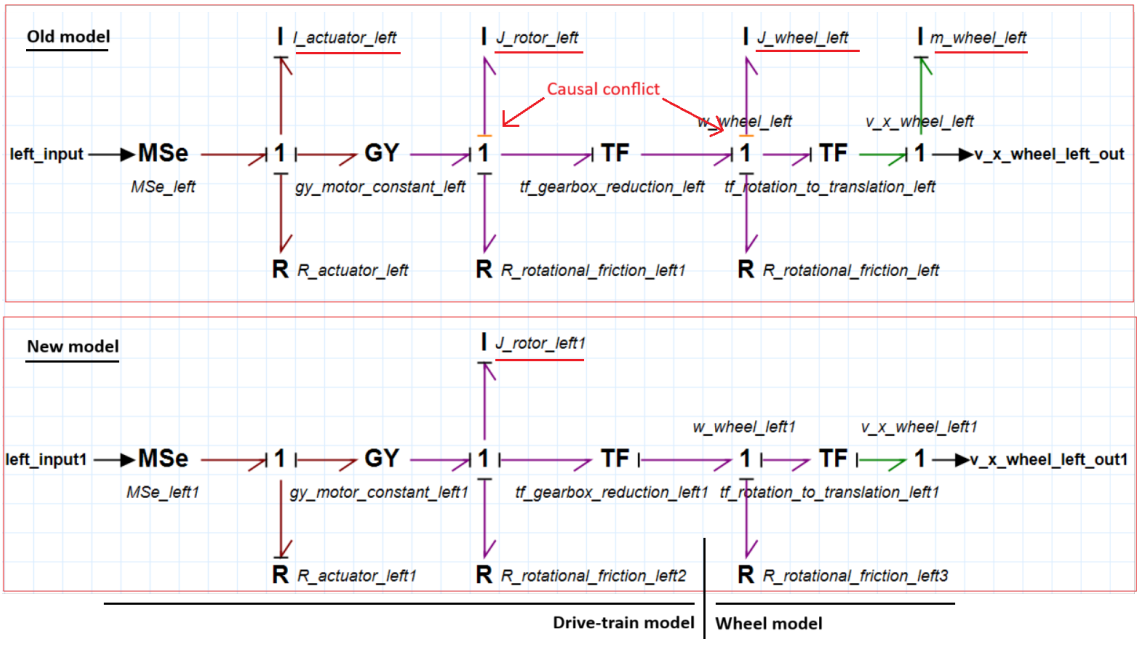

Figure 6.5: Adjustments to the drivetrain and wheel models

6.3.2 Spring damper connections

The moment of inertia of the body, the mass of the body and the wheels all have preferred flow-out-causality meaning that they cannot all have preferred causality when they are connected with a zero-junction. Precisely one connection of a zero-junction must have effort-out causality. This is shown in Figure 6.6.

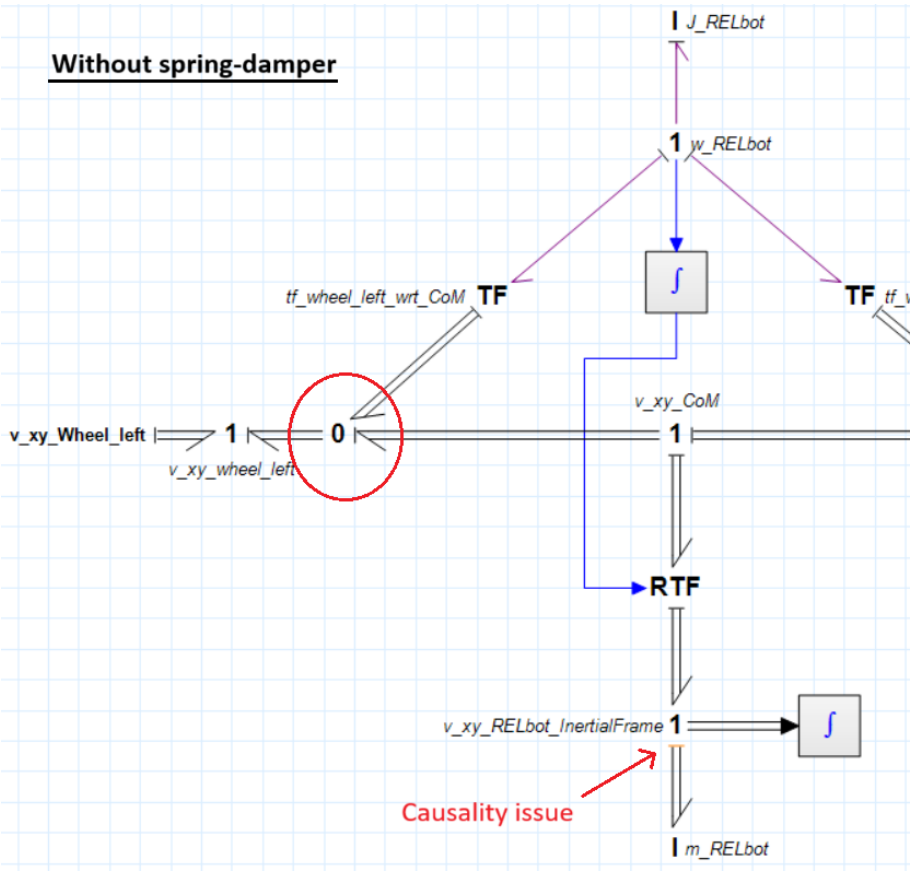

Figure 6.6: Left part of the rigid-body model without spring-damper

Step 4 of the user guide suggests three possible methods to solve this: remove non-essential elements, combine elements, or add elements. There are no non-essential elements that can be removed to solve this. The connections are too complex to combine elements. This means elements need to be added to solve the causality issue. A spring element, represented by a C, can be added to the zero-junction as this has preferred effort out causality. A resistor is added to damp the oscillations that occur as a result of the spring as shown in Figure 6.7.

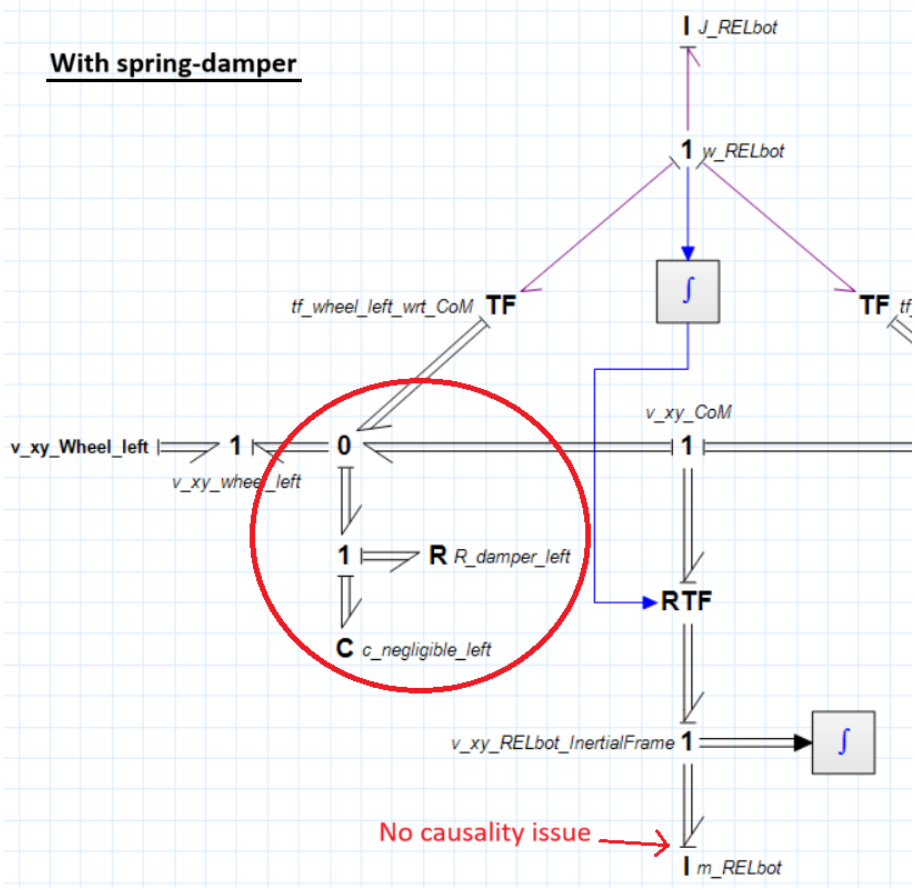

Figure 6.7: Rigid body model with spring-damper

Adding a spring-damper to connect multiple I-type elements is a good way to solve this issue (Broenink, 2020, Chapter 2.6.3). The spring applies a force to the masses to keep the position difference between the points small while the damper reduces oscillation. A weak spring leads to big position differences while a stiff spring leads to high frequency oscillation and possible instability as mentioned in the user guide. The R-element damps the oscillation. The spring-damper models elasticity in the mechanical connection that was modeled as rigid previously. The spring-damper values are selected to keep position difference small while avoiding high frequency dynamics. Plotting the integrated flow of the spring gives insight into the displacement in the connection.

6.3.3 Rigid body to wheel connection

The Rigid-body uses 2D-bond-graph elements since it can move in the X and Y plane but the wheel models are 1D-bond-graph elements only moving in the Y plane. This was solved in the previous model with a resistance element and a powermux component which introduced an algebraic loop.

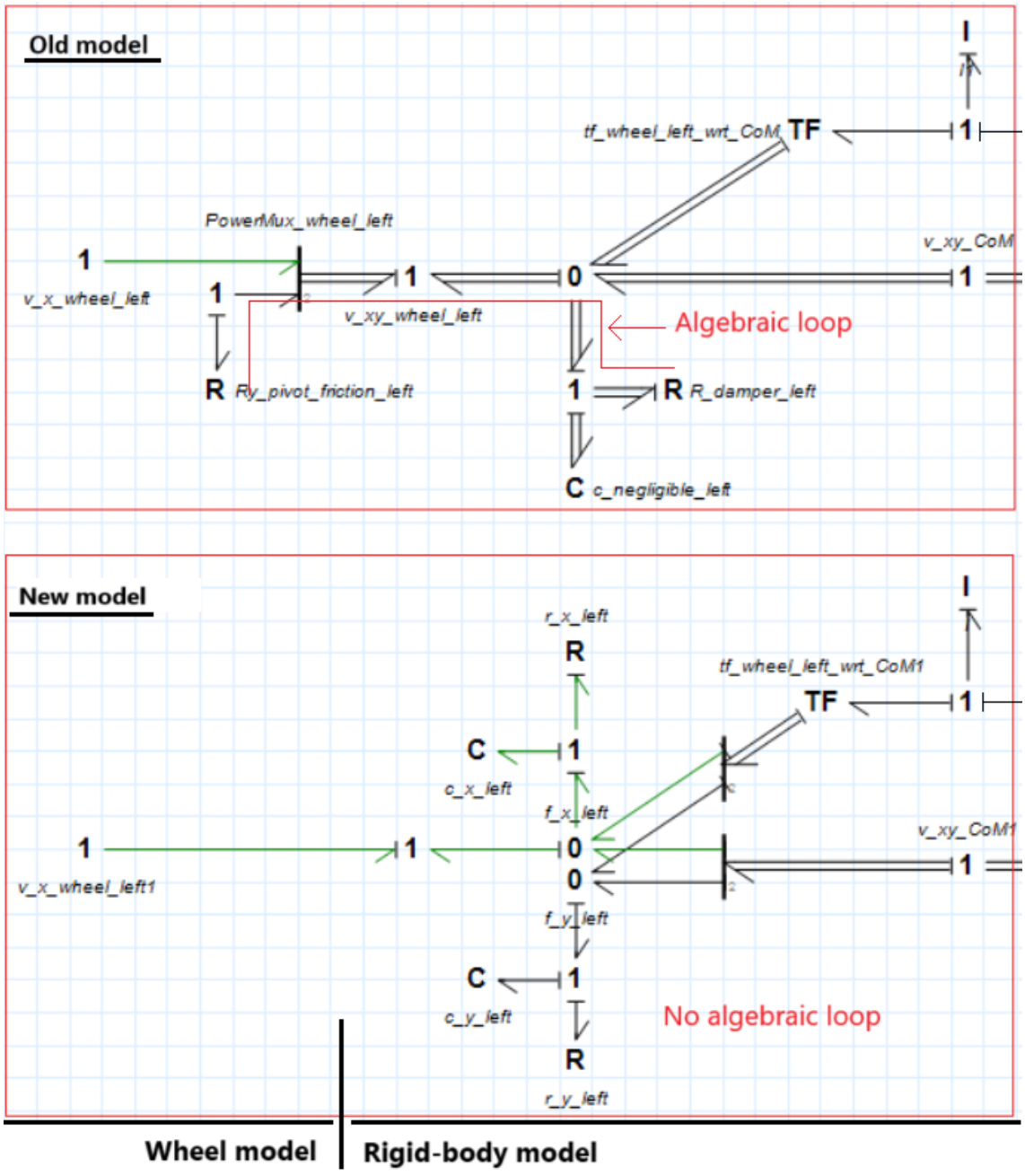

Figure 6.8: Wheel to rigid body connection adjustment

The new model splits the 2D rigid body into separate X and Y sections. Since the wheels can only be actuated in the X direction they can be directly connected to the X position elements of the Rigid body. This means the R pivot friction can be removed and this resolves the algebraic loop.

6.4 3D Visualisation

3D visualisation helps to better understand how the plant behaves compared to standard value plots. In 20-Sim a 3D model can be added and its position can be linked to certain variables of the simulation model. Then after simulation the 3D visualisation render can be played. The following aspects are important for meaningful results.

1. **Consistent coordinate system**
   A well defined coordinate system in the 20-Sim model and the 3D model helps to avoid confusion in the 3D visualisation process.
2. **3D model alignment**
   The offset of the 3D model to the origin of the design space determines the offset to the origin in the 3D render. The 3D model is centered on the origin to simplify 3D rendering in 20-Sim.
3. **Scaling**
   Scaling is used to make the 3D model fit well inside the visualisation space. There are various points where scaling can be applied. Scaling must be consistent between all parts to ensure correct results. The parts can be scaled as a group to ensure that the 3D visualisation is not too small or too big for the visualisation space, and this way the scaling between parts is unchanged.

Figure 6.9 shows the 3D RELbot model in 20-Sim. The 3D model of the RELbot consists of a body and two wheels. Since the body and wheels always have the same relative position to each other they are put in the same reference frame and they move as a group.

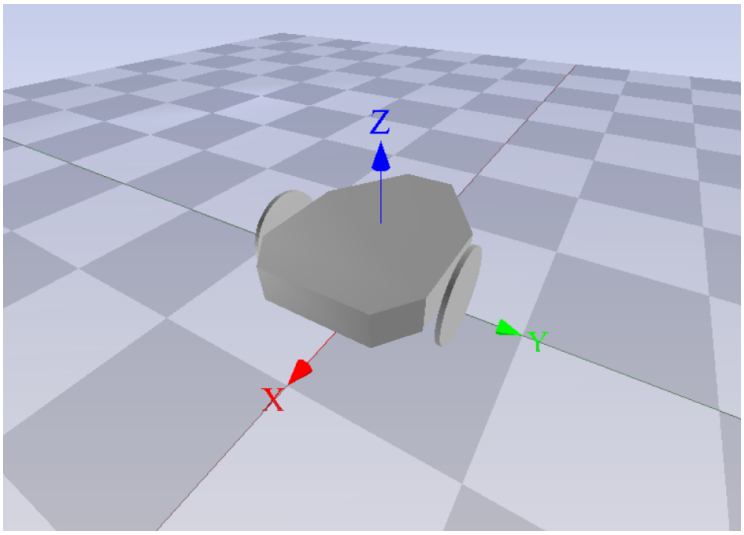

Figure 6.9: 3D visualisation of the RELbot in 20-Sim.

6.5 Implementation of models on the HiL simulation platform

6.5.1 Input and output processing

The controller with a sample frequency of 1 kHz is implemented with the Xenomai framework on the ECS. The encoder reader on the FPGA of the ECS counts the encoder pulses and provides the encoder count to the ECS where it is converted to meters and given to the controller model. The output of the controller is a duty-cycle between -1 and 1 but the datatype for the PWM generator is an integer value between -2048 and 2047 so the duty-cycle is scaled before it is sent to the PWM generator on the FPGA.

The plant model is implemented with the Xenomai framework on the Digital Twin. The PWM reader on the FPGA provides the PWM value between -2048 and 2047 which is scaled down with the PWM resolution to -1 to 1 before it is given to the plant model. The output of the plant is the wheel positions in meters which is converted to the relative encoder position before it is sent to the encoder emulator. The code for these conversions for the Digital Twin is shown in Appendix G.

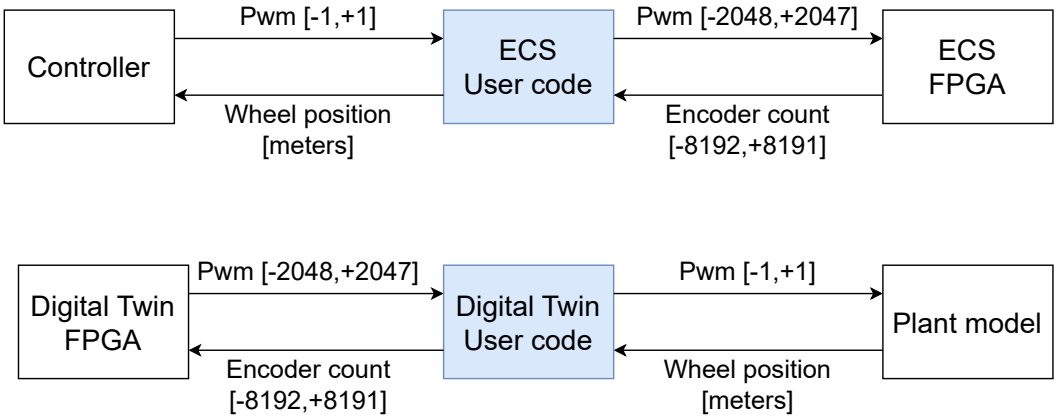

Figure 6.10: Input and output conversions between the FPGAs and the models

6.5.2 Selecting the update frequency of the Digital Twin

Heuristic approach

The heuristic approach described in Section 3.8.2 sets the Digital Twin update frequency between 4 and 10 times higher than the ECS update frequency. The update frequency of the ECS is 1 kHz so the recommended frequency is between 4 kHz and 10 kHz.

Performance based approach

At the time of the performance-based approach measurements we did not set the CPU in performance mode because we were unaware of this functionality. Consequently, the measurements of the theoretical maximum frequency are incorrect and the theoretic maximum frequency is higher than stated here. This oversight does not affect the final choice of the update frequency.

The performance measurements based approach described in Section 3.8.2 calculates a theoretic maximum frequency and uses this to give a good initial update frequency for the Digital Twin. The overhead of the firm real-time loop is 35 μs (Raoudi, 2024). The average calculation time is found through testing. The model calculation function is executed many times and the average duration is measured. The average time is 85 μs from experiments. 1/120 μs = 8.3 kHz as theoretical maximum frequency. This approach recommends an update frequency of 75% of the theoretical maximum, which is around 6 kHz.

Chosen update frequency

The approaches suggest an update frequency of 4 kHz to 6 kHz as a starting point for testing. Multiple tests are done at different frequencies to find the average error and the missed deadlines. With 4 kHz the average error is very small and there are very few missed deadlines. The marginal performance improvements at higher frequencies are not worth the increase in computational resources. As a result 4 kHz is chosen as the update frequency for the Digital Twin of the RELbot.

6.6 Live visualisation

The choice is made to implement live visualisation of the X and Y position of the centre-of-mass of the RELbot so the X and Y position are added as outputs of the model. In the implementation on the Digital Twin, a custom ROS message is added to send the centre-of-mass position to a ROS topic. To visualise the position live, a computer with the following components is required:

1. ROS.
2. A connection to the network where the ROS message is published.
3. A ROS visualisation package.
4. A display.

The Raspberry Pi of the Digital Twin meets these requirements so it is used to do the visualisation. The visualisation package plotjuggler runs on the Raspberry Pi and the visualisation window is displayed on a connected computer.

The visualisation plot shown in Figure 6.11 displays the last data point and a number of points before it, which is why the plot shows a line instead of a single dot.

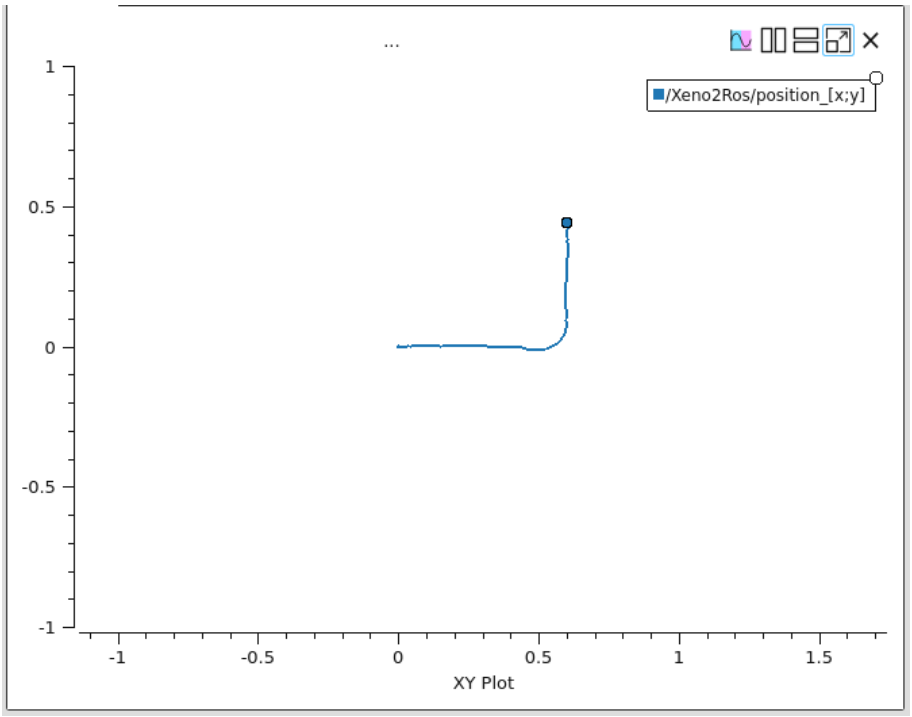

Figure 6.11: Live visualisation of the Digital Twin using Plotjuggler

7 Case study Testing

7.1 Introduction

This chapter contains the tests of the case study and their results. In Chapter 5 the Digital Twin is compared to the plant model in 20-Sim to look at the hardware platform rather than the specific application. In this chapter the Digital Twin is compared with the real RELbot in order to test the performance of the Digital Twin and the quality of the model. The data from the HiL simulation is green in the plots, the data from the real RELbot is purple in the plots and the difference between the two is plotted in red.

7.2 Digital Twin and RELbot closed-loop test

7.2.1 Goal and setup

The goal of this test is to see if the theoretical controller developed in 20-Sim that is implemented on the ECS can control the Digital Twin and the RELbot. A controller test in 20-Sim is documented in Appendix E. The controller is provided with a setpoint input profile and connected to the Digital Twin and the output is logged. Then the test is repeated with the real RELbot and the output is logged. The logged data files are put in 20-Sim for comparison. The output is compared to the setpoint to find the position error. The test setup is shown in Figure F.4.

7.2.2 Measurements

The top plot of Figure 7.1 shows the wheel position setpoint and the wheel position output of the Digital Twin and the wheel position output of the RELbot. The lines overlap so only the purple line is visible. The bottom plot shows the error between the setpoint and the measured value for both the Digital Twin and the RELbot.

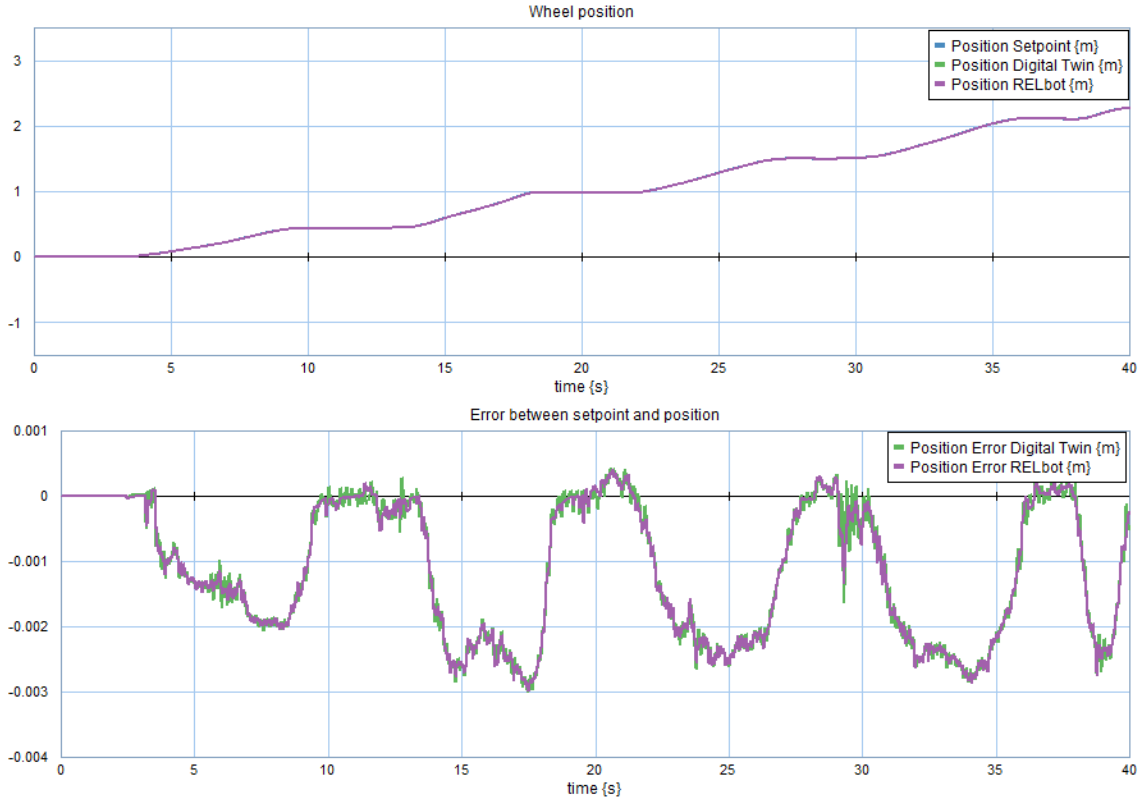

Figure 7.1: Closed-loop controller tests of Digital Twin and RELbot.

7.2.3 Interpretation

The error between measured position and setpoint position is close to zero for both as during the 2.5 m test the error stays below 3 mm. This shows that the controller works well for both the Digital Twin and the RELbot. The error plot of the Digital Twin looks very similar to the error plot of the RELbot which isexpected since they should have very similar dynamics.

7.3 Digital Twin vs RELbot open-loop comparison

7.3.1 Goal and setup

The goal for this test is to see how similar the Digital Twin is to the RELbot in an open-loop test. This test is used to find the average open-loop error and it is also used to calculate the credibility assessment score. The ECS of the HiL simulation setup is copied to the ECS of the RELbot which has identical hardware. The RELbot ECS and the ECS of the HiL simulation setup are given the same input profile of PWM values. The encoder output of the RELbot is logged and the encoder output of the Digital Twin is logged. The logged data is compared and plotted. The open-loop setup in 20-Sim is shown in Figure F.3.

7.3.2 Measurements

The top plot of Figure 7.2 shows the measured position value of the Digital Twin and of the RELbot. The bottom plot shows that the difference between both position values is less than 4.6 cm during this test.

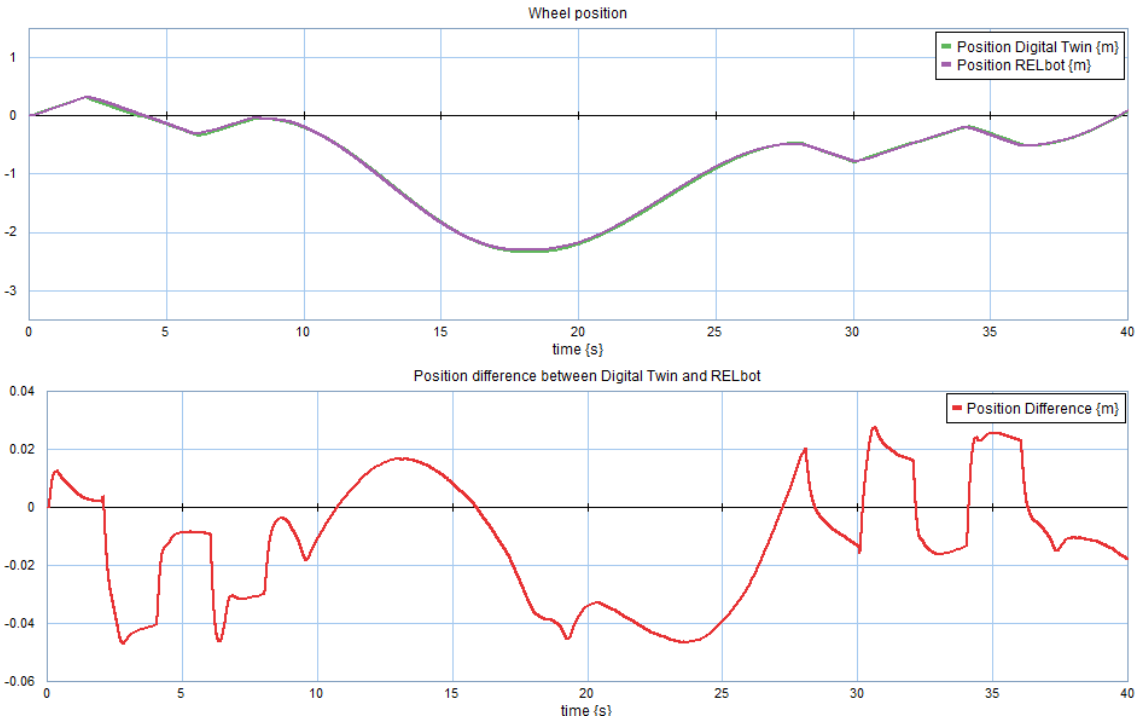

Figure 7.2: Open-loop controller test of Digital Twin and RELbot.

7.3.3 Interpretation

The test shows that the Digital Twin is quite similar to the RELbot but it does have a measurable difference in the output. This is expected since the Digital Twin uses a simplified representation to model the dynamics of the RELbot. The average error is calculated from the plot data and is found to be 24.4 mm which corresponds to 1 % which is below the required 5 % so it meets the

accuracy requirement. This test shows that the Digital Twin works well and the RELbot model is good enough.

7.4 Credibility assessment of the RELbot

7.4.1 Goal and setup

The goal is to find the credibility score of the Digital Twin by comparing Digital Twin output data with output data from the real RELbot. Both are given the same PWM input profile. The time-domain credibility assessment method is used on the data of the top plot of Figure 7.2. The passing score is set at the recommended value of 60 % meaning that scores above 60 % are good enough and scores below 60 % are insufficient. The confidence interval is set at the recommended 5 %.

7.4.2 Result

Parameters:

$K_p = 0.05$, $n_{pass} = 0.6$

$K_e = \frac{n_{pass}}{\sqrt{1-n_{pass}^2}} = 0.75$

Measurements:

Error threshold: $\epsilon_t = 0.1311\ m$

Difference between HiL and RELbot curve: $e_t = 0.0244\ m$

$\eta_{time} = \frac{K_e * \epsilon_t}{\sqrt{(K_e * \epsilon_t)^2 + e_t^2}}$

$\eta_{time} = \frac{0.75 * 0.1311}{\sqrt{(0.75 * 0.1311)^2 + 0.0244^2}} = 0.9706 = 97\ \%$

7.4.3 Interpretation

The credibility assessment score is calculated to be 97 %. This score is close to 100 % which means that a good Digital Twin has been developed. It also means that the requirement of a sufficient credibility score for the Digital Twin has been achieved.

8 Conclusions and Recommendations

8.1 Conclusions

HiL Simulation infrastructure development and testing
A HiL simulation infrastructure was developed and tested. The infrastructure consists of user guides and extensions of the Xenomai framework. The tests show that the requirements are met and that the infrastructure can be used to develop a Digital Twin and perform HiL simulation. The hardware platform of the Digital Twin introduces a negligible error of 7×10^{-3} % compared to the 20-Sim simulation which is below the 1 % requirement. This proves that the Raspberry Pi with Icoboard FPGA and the Xenomai framework is a suitable platform for HiL simulation.

RELbot Digital Twin development and testing
The goal of developing and testing a Digital Twin of the RELbot with live visualisation functionality was achieved. The Digital Twin simulates the RELbot in real-time on a Raspberry Pi. An open-loop output comparison between the Digital Twin and the real RELbot shows that the average error is 1 % which is below the 5 % requirement. This means that the goal of developing an accurate real-time model of the RELbot has been achieved. 3D visualisation of the RELbot has also been successfully implemented. The tests show that the requirements are met and a good Digital Twin is developed.

Credibility assessement
The goal of performing credibility assessment was achieved. A credibility assessment script is developed which determines the credibility score based on simulation data. The credibility assessment of the Digital Twin gave a score of 97 %. A score of 60 % is sufficient so 97 % is certainly good enough. This means that the Digital Twin simulation results have high credibility and can be used to test Embedded Controllers with.

8.2 Recommendations

Improve the RELbot reference frames selection
The wheels should be modeled as rigid-bodies with their own reference frame. In the current model the wheels are modeled in the same reference frame as the body of the RELbot. With their own rigid-bodies the axes of rotation of the wheels are modeled the same way as is expected of the real robot with positive rotation corresponding to the right-hand-rule.

Quality-of-Service check
Expand the Digital Twin to perform a Quality of service (QoS) check at the end of a simulation that evaluates the missed deadlines and informs the user if the test results are valid or not. If a HiL simulation test gives bad results due to many missed deadlines a user might think the controller is bad and change it. Missed deadlines reduce Digital Twin reliability. It is crucial to define acceptable QoS metrics tailored to the specific application's needs and constraints and it would be nice to have the QoS check done automatically. The pattern of missed deadlines is also important so QoS for firm real-time systems is often done with a sliding window analysis. (Donglin Xiu, Hu, Lemmon and Qiang Ling, 2003; Hamdaoui and Ramanathan, 1995)

Improve live visualisation
Currently the visualisation package runs on the Digital Twin which renders the live plot and sends this to a computer via SSH. This is a suboptimal solution as this puts the computational load on the Digital Twin. As a result, the live visualisation becomes laggy when the windowsize of plotjuggler is too big. A better solution would be to subscribe to the ROS topic from a computer and run the visualisation package on the computer.

Expand live visualisation for the RELbot
It is recommended to expand the live visualisation of the RELbot to include the angle of the body as well. The current visualisation only displays the X and Y position of the centre of mass. If this is not possible with plotjuggler it means a new visualisation package is required. It also means that the angle of the RELbot must be added as an output in the model and added to the ROS message.

Tests with different plants
More research can be done to find the limitations of the HiL simulation infrastructure when it comes to high-frequency plant dynamics, higher-order systems or different types of systems. A setup with a robot arm could be interesting because this also requires implementation of end-stops which are discrete events which have not been part of the HiL simulations so far. This would also require using more states of the state machine which was not required for the RELbot case-study.

A User guide for real-time capable modeling

Steps for real-time capable and explicit model development

1. **Model simplifications:**
   Simplify large-scale or complex systems to reduce computation time and meet deadlines by order reduction, linear approximation of non-linear dynamics, and removal of non-essential dynamics.
2. **Avoid high stiffness dynamics:**
   High stiffness dynamics introduce high frequency oscillations which require very small time steps for stability.
3. **Avoid dynamic causality changes:**
   Explicit methods may fail if the causal assignment changes during operation due to, for example, switching power electronics or mechanical contact.
4. **Ensure preferred causality:**
   Explicit models have preferred causality for all energy storage elements.
   1. Remove non-essential elements.
   2. Combine elements.
   3. Introduce extra elements.
5. **Remove algebraic loops:** Explicit models have no algebraic loops.
   1. Remove non-essential elements.
   2. Combine elements.
   3. Introduce extra elements.
6. **Choose an explicit integration method:** Euler, Runge-Kutta 2 or Runge-Kutta 4. Preliminary tests show that Runge-Kutta 4 has the most accurate results. Runge-Kutta 4 should be the default.

Based on bond-graph books such as Broenink (2020).

Step 1: Model simplification

In model development in general, but especially for real-time simulation, the model must be as simple as possible while still capturing the important dynamics of the system. This way a competent model can be developed. Each additional independent energy storage element increases the number of states of the model and the order of the model. In general, more states increase the calculation time of the simulation or control algorithm, because there are more differential equations to solve and more variables to track.

To verify if an element can be removed a simulation with the element and without the element can be done. If removing an element or section of the model has a negligible effect on the simulation outcome then it might not be necessary to include this part in the model. Make sure that the simulation that is used to measure the influence of an element actually excites the relevant part of the model. That is to say: measuring the influence of a mass moving at a constant velocity does not make sense. Perform a simulation with acceleration and deceleration instead.

Step 2: Avoid high stiffness dynamics

A stiff spring introduces high frequency oscillations. The natural frequency of a mass-spring system is given by:
$f_n = \frac{1}{2\pi}\sqrt{\frac{k}{m}}$
This shows that a spring with high stiffness k results in a high natural frequency. Note that the electrical equivalent of a spring is a capacitor where the stiffness corresponds to the inverse of

the capacitance. A capacitor with a very small capacitance introduces high frequency oscillations.

To avoid high frequency dynamics each stiff springlike element should be considered. If there is a very stiff spring in the model then it might be better to model it as completely rigid. If this spring connects two I type elements these elements could be modeled as a single I element. This removes two storage elements and a stiff spring.

Step 3: Avoid dynamic causality changes

Maintaining stable causality is essential for ensuring the efficiency and stability of models. Dynamic causality changes can result in causality issues during simulation. Avoiding dynamic causality changes ensures that the simulation remains straightforward, predictable, and computationally feasible. Below are some common causes of dynamic causality changes:

1. **Discrete events**
   In systems that include discrete events or sudden state changes dynamic causality changes can occur. For example, a switch in a power circuit, a mechanical system that engages or disengages or mechanical collisions.
2. **Switching control strategies**
   In certain models the control strategy may change dynamically based on inputs or certain conditions. For example, a system might switch from a PID controller to an adaptive controller.
3. **Nonlinear systems**
   Certain models may operate linearly under certain conditions but switch to a nonlinear model under other conditions, changing the causal relationships.
4. **Conditional logic**
   If the model uses if-else branches or similar logic it can lead to dynamic causality changes.

Step 4: Ensure preferred causality

To solve a causal conflict the model must be changed. Sometimes it is necessary to restructure part of the model and change multiple aspects, but often one of the following adjustments solves the issue:

1. **Remove non-essential elements**
   If there is a causal conflict and one of the elements is non-essential then it can be removed to solve the issue.
2. **Combine elements**
   If two elements have a causal conflict sometimes they can be taken together to eliminate the conflict. This can be performed via transformations in the graph. The complexity of this operation depends on the size and kind of submodels along the route betweeen the two elements.
3. **Introduce elements**
   This is the third option since this increases the complexity of the model. Sometimes removing elements and combining elements can not solve a causal conflict and additional elements are required. The added elements can be parasitic, for example, to add elasticity (C-elements) in a mechanical connection, which was modeled as rigid. Additionally adding a damping element reduces the simulation time considerably, which is being advised. (Broenink, 2020, Chapter 2.6.3).

Step 5: Remove algebraic loops

Algebraic loops occur when two resistive elements are in conflict. Often this is an indication that a storage element was not modeled, which should be there from a physical systems viewpoint. The three steps are the same as in step 4.

Step 6: Choose an explicit integration method

The 20-Sim C++ code generation tool allows three explicit integrators:

1. Euler
2. Runge-Kutta 2
3. Runge-Kutta 4

Runge-Kutta 4 is recommended.

B User guide for HiL simulation implementation

1. **Code generation**
   Explains how to generate code from the explicit model.
2. **Generated code implementation**
   Explains how to implement the generated code in the framework.
3. **Simulation parameters**
   Explains how to adjust the simulation time-step, the simulation duration, and the integration method.
4. **User implementation**
   Explains what the user has to implement for the parts that are specific to the use case. This is pre- and post-processing of the model inputs and outputs, executing the calculation, and optionally logging.
5. **Output analysis**
   Explains how the logged data from the ECS can be processed and plotted using 20-Sim. Explains how the credibility assessment script is used. The credibility assessment script is shown in Appendix C.
6. **Live visualisation**
   Explains how to adjust the model for live visualisation, how to use ROS topics to communicate the data, and how to do live visualisation.
7. **From HiL simulation to the real setup**
   Explains the steps the user should take to connect the ECS to the real plant.

1. Generate code from explicit model

1. In 20-Sim, click `Start simulator` to go to the simulation window.
2. Click `Run Properties` to go to the simulator settings.
3. Select an explicit integration method. We recommend `Runge-Kutta 4`.
4. Click `Set Properties`, and set the step-size. See Section 3.8.2 for step-size recommendations.
5. Click `Run simulation` to run the simulation.
6. Click `Tools`, `Real Time Toolbox`, `C-Code Generation`.
7. Select C++ class for 20-sim submodel.
8. Under `Submodel:`, choose a (sub)model from your model that you want to generate C++ code from. This (sub)model should be called LoopController. Calling it LoopController ensures that the user does not have to make many adjustments in the CMake file in C++.
9. Under `Output Directory:`, choose a location to store the model in. It must be stored in a folder named `plant`. This is also to make sure the user does not have to make many adjustments in the CMake file in C++.

2. Implementation of the generated code on the Digital Twin

The implementation for the Digital Twin is very similar to the implementation of the ECS which already has a user manual written by Raoudi (2024, Appendix F.2). The user guide below explains which steps are different and what steps are missing. The ECS implementation guide has 31 steps. The adjustments are summarised as follows. `controller` in the manual of the ECS should be `plant` for the Digital Twin. This is because the controller model is saved in a folder named `controller`, while the plant is saved in a folder named `plant`. The name `LoopController` should not be changed however.

5: Use Template-20sim-Inverse.

8: Put the data for the live visualisation in the Xeno2Ros message.

9a: Go to the /plant folder.

16: (Notice) The state machine is not as important for the Digital Twin. When the Digital Twin is started it goes to the run state automatically. You don't have to do anything here.

18: The variable `sample_data` provides all the sampling data from the FPGA of the Digital Twin and has the following structure:

```
struct IcoRead
{
    int channel1;       ///< Pwm value from channel 1
    int channel2;       ///< Pwm value from channel 2
    int channel3;       ///< Pwm value from channel 3
    int channel4;       ///< Pwm value from channel 4
    bool channel1_1;    ///< Digital input 1 from channel 1
    bool channel1_2;    ///< Digital input 2 from channel 1
    bool channel2_1;    ///< Digital input 1 from channel 2
    bool channel2_2;    ///< Digital input 2 from channel 2
    bool channel3_1;    ///< Digital input 1 from channel 3
    bool channel3_2;    ///< Digital input 2 from channel 3
    bool channel4_1;    ///< Digital input 1 from channel 4
    bool channel4_2;    ///< Digital input 2 from channel 4
};
```

The variable for controlling the FPGA output of the Digital Twin is called `actuate_data` and it has the following structure:

```
struct IcoWrite
{
    int16_t     enc1;       ///< Encoder value for channel 1
    bool        val1;       ///< Digital output of channel 1
    int16_t     enc2;       ///< Encoder value for channel 2
    bool        val2;       ///< Digital output of channel 2
    int16_t     enc3;       ///< Encoder value for channel 3
    bool        val3;       ///< Digital output of channel 3
    int16_t     enc4;       ///< Encoder value for channel 4
    bool        val4;       ///< Digital output of channel 4
};
```

19: Implement the computation.

- **20-sim variant**
  1. The 20-sim model is already imported and is called `plant`. Through the `plant.calculate` function one computation step is performed. The `plant.finished` function tells when the simulation is finished. Explore the plant class for other functionality provided by the 20-sim generated code.

31 This is not necessary on the Digital Twin.

3. Adjusting simulation parameters

Simulation parameters can be adjusted in the LoopController.cpp file. This file is found in the following location: `workspace/src/<user-application>/Xenomai/plant`. Open the `LoopController.cpp` file. The simulation duration can be set at the the `m_finish_time` variable. The simulation time-step size can be set in the `m_step_size`

variable. The integration method can be adjusted in the LoopController.hpp file. The class of the object called `myintegmethod` can be changed to one of the classes defined in the `xxinteg.h` file.

4. User implementation

There are four main steps that the user has to do to implement the generated code with the software framework on the Raspberry Pi as shown in Figure B.1.

1. Processing of the measured input data of the FPGA so it can be used by the model calculation.
2. Calculating the new outputs of the model.
3. Processing of the model calculation output values so they can be sent to the FPGA in the next cycle.
4. Logging the input and output values. This is optional on the Digital Twin side but is definitely recommended on the ECS side.

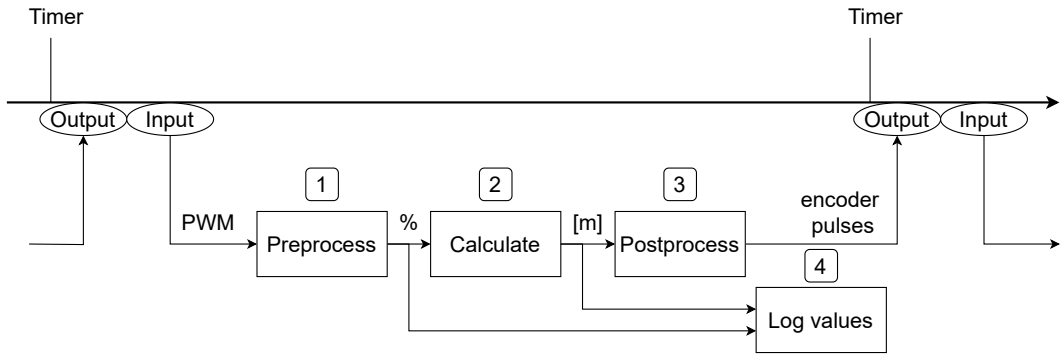

Figure B.1: Overview of user implementation of code

4.1. Preprocessing

The PWM value from the FPGA is an integer between -2048 and +2048. If the input of the model is a duty-cycle, the PWM value should be divided by the PWM resolution. If the input is a voltage, the duty-cycle should be scaled accordingly.

4.2. Calculate

Before running the model calculation, correctly put the input values in the array `u[]`. To run the model, use `LoopController.Calculate(u,y);`. This will update the outputs in the array `y[]`.

4.3. Postprocessing

The output of the model may be meters, or radians or something else. This should be converted to an encoder position before it can be sent to the FPGA. Convert the position to an integer datatype. The icoboard_inverse class applies the modulus of the encoder resolution to the given encoder position to get the relative encoder position.

4.4. Logging

Logging should be done on the ECS side and is optional on the Digital Twin side. This is because tests with the real plant are logged on the ECS side so to compare the two it is best to do logging on the same device for both tests.

Put the variables that should be logged in the logger. The user manual for this is provided by Raoudi (2024, Appendix D). Below is an example where the duty-cycle and position of two wheels is logged. The application is called Demo. In the Demo.hpp file:

```
#pragma pack (1)
struct ThisIsAStruct
{
    double dutycycle_right = 0.0;
    double position_right = 0.0;
    double dutycycle_left = 0.0;
    double position_left = 0.0;
};
#pragma pack(0)
```

In the constructor of the Demo class:

```
logger.addVariable("dutycycle_right", double_);
logger.addVariable("position_right", double_);
```

```
logger.addVariable("dutycycle_left", double_);
logger.addVariable("position_left", double_);
```

In Demo::initialising():

```
logger.initialise();}
```

In Demo::run():

```
data_to_be_logged.dutycycle_right = u[1];
data_to_be_logged.dutycycle_left  = u[0];
data_to_be_logged.position_right = y[1];
data_to_be_logged.position_left  = y[0];
```

5. Output analysis

5.1 Sourcing logged data to 20-Sim

It is advised to use logged data from the ECS. This is because the ECS is used for both the Digital Twin and the real plant so measuring at the same point ensures only one variable is changed at a time between tests. The logged data can be converted to a .mat file using the logger decoder made by Raoudi (2024). This can be found in the following folder:

```
ros2-xenomai4-framework/XenoRosFramework/Common/
XenoFrtLogger_decoder
```

Use the following command: `python binairy_to_matlab.py`.

To compare the HiL simulation with the 20-Sim simulation or with data from the real plant, the data can be sourced to 20-Sim using CSV-files. 20-Sim interprets the first column as time-data, so a column of time-data has to be added for example in Matlab. To add the logged data with the time-column to 20-Sim, use the `DataFromFile` block found in: Library/Signal/-Sources/DataFromFile. Open the `DataFromFile` block and provide the path to the CSV-file.

5.2 Performing credibility assessment

The following steps explain how to perform Time-Domain Credibility Assessment.

1. Source the data from the experiment with the Digital Twin and the experiment with the real plant to 20-Sim.
2. Plot the data in a plot like Figure B.2.
3. Right click on the plot. Go to `Save Data To File` and click `Only for this plot`.
4. Save the CSV file.
5. Open the Credibility Assessment script in Matlab.
6. Import the CSV file in Matlab.
7. This will create a variable in the Matlab workspace of the type: table. This variable will have multiple columns: The first column is time-stamp data, the other columns are the curves of the plot.
8. Put the name of the CSV file from the previous steps in the `table2array` function at the top of the Matlab script.
9. In line 10 and 11 the two variables y_s and y_e should correspond to the data of the two curves that we want to compare. The first curve in the plot is the second column of the variable in Matlab and the second curve is the third column.
10. Run the script, the credibility score is stored in the nt variable. The score is between 0 and 1. Multiply by 100 to get the score as a percentage.

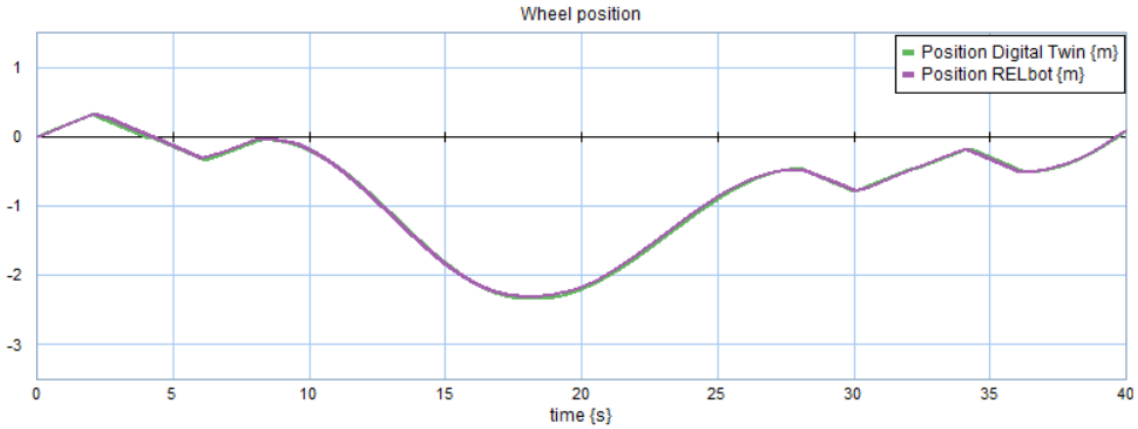

Figure B.2: Plot for credibility assessment

6. Live visualisation of an XY-plot using SSH and Plotjuggler

This section explains live visualisation using SSH. In this setup the ROS Plotjuggler package is executed on the Digital Twin and the data is sent to a computer which is connected through SSH.

1. Add the variables to be visualised as outputs to the plant model in 20-Sim, generate the code and copy the generated code to the C++ workspace.
2. The variables are outputs of the `plant.calculate()` function and are stored in the array `y`.
3. Add the variables to the `Xeno2Ros` message (probably called `xeno_data`). The default datatype of the 20-Sim model output is a double. To use this directly, the variables in the `Xeno2Ros` message datatype should be `float64`.
4. Connect a PC to the Digital Twin using:

```
ssh -X <rpi-name>@<ip\_address>
```

5. Start the HiL simulation.
6. Start the Ros-Xeno bridge using:

```
ros2 run ros_xeno_bridge RosXenoBridge
```

7. Start X-Launch on the PC that is connected to the Digital Twin.
8. Ensure that the Raspberry Pi displays visuals on the computer using:

```
export DISPLAY=<pc_ip_address>:0.0
```

9. Start the visualisation package on the Digital Twin using the following command in a terminal:

```
ros2 run plotjuggler plotjuggler
```

10. This should open a plotjuggler window on the connect PC.
11. In the lefthand pane click `Start`.
12. Select the variables that should be plotted from the list and press `Ok`.

13. The variables now appear on the left side. Select the two variables for the XY-plot and drag them with the right-mouse-button to the plot window to create an XY-plot. Dragging them with the normal left-mouse-button will plot the two variables against time instead of agains eachother.
14. Make sure the `1:1` button in the top right is selected to ensure the XY-plot is in a 1:1 ratio.
15. Adjust the buffer size to display more or fewer past data points.
16. To adjust the min and max X values of the plot, right-mouse-click on the plot and click `edit curves`. The limits can be adjusted there.
17. Note that the window size of plotjuggler heavily impacts the performance. Keep the window small to keep the visualisation running smoothly.

7. From HiL simulation to the real plant

If the HiL simulation results are satisfactory, the plant behaves as expected and the controller works well, then the ECS can be tested with the real plant. In some cases it may be possible to disconnect the wires from the ECS to the Digital Twin and connect them to the real plant but this is not always the case. Sometimes the ECS is built in to the real plant and it cannot be disconnected. Assuming the ECS hardware connected to the real plant is identical to the ECS connected to the Digital Twin, the whole code workspace can be copied to the ECS of the real plant. If a git repository is used it can be cloned to the ECS. Built the code, source, and run. Always consider safety before testing.

C Credibility assessment script

```
CredibilityData = table2array(Test8HilData);

y_s = CredibilityData(1:end,2); %Simulation data
y_e = CredibilityData(1:end,3); %Experimental data
timestep = CredibilityData(2,1);
%% Time domain

%Confidence interval
Kp = 0.05;
n_pass = 0.6;

Ke = n_pass/sqrt(1-power(n_pass,2));

eta_t = Kp * (max(y_e)-min(y_e)); %Error threshold

%Average error = sqrt(1/n sum (difference^2)
n = size(y_e,1);
difference = y_e-y_s;
differenceSquare = power(difference,2);

%Average error
e_t = sqrt(1/n*sum(differenceSquare));

%Credibility assessment score
nt = Ke*eta_t/(sqrt(power(Ke*eta_t,2) + power(e_t,2)))
```

D 20-sim model

Full bondgraph shown in Figure D.1. It shows the controller, the plant, the filedata containing HiL simulation data and a compare block which calculates differences between signals.

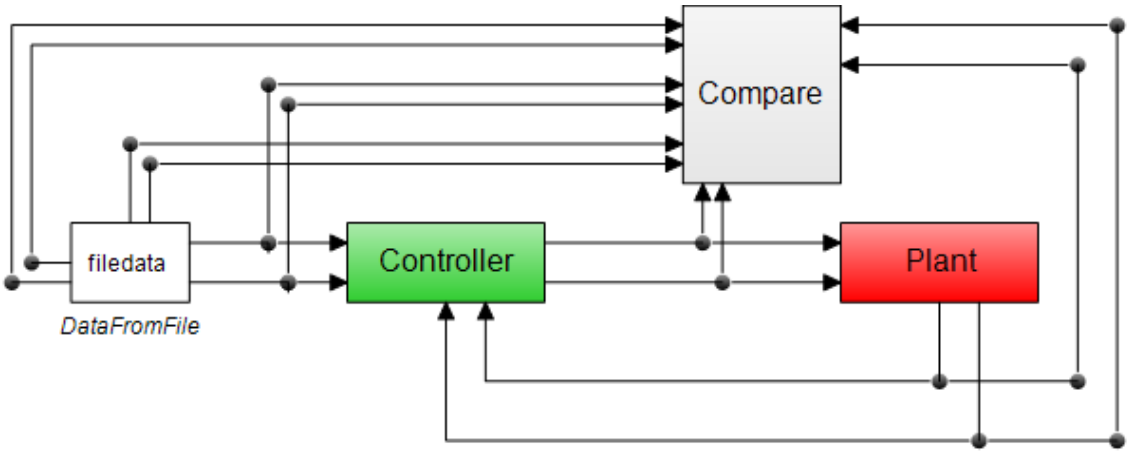

Figure D.1: Full bondgraph

Full plant subsystems are shown in Figure D.2.

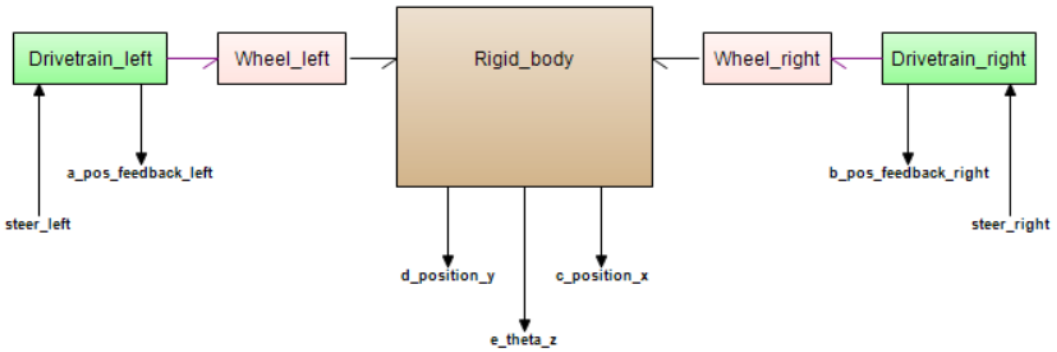

Figure D.2: Full plant bondgraph

The left drivetrain model is shown in Figure D.3.

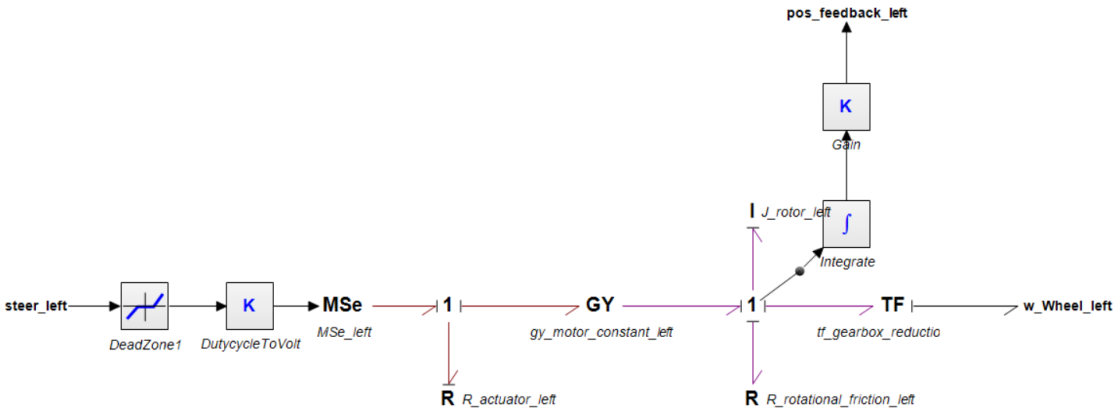

Figure D.3: Left drivetrain bondgraph

The left wheel model is shown in Figure D.4.

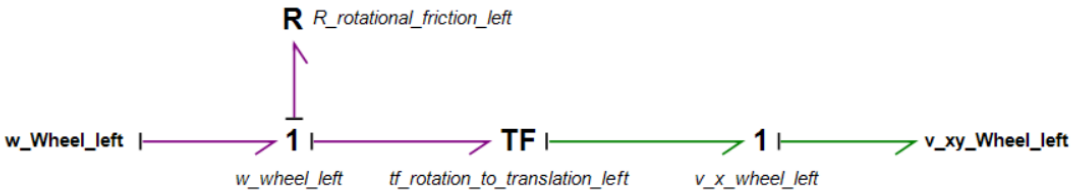

Figure D.4: Left wheel bondgraph

The Rigid body model is shown in Figure D.5.

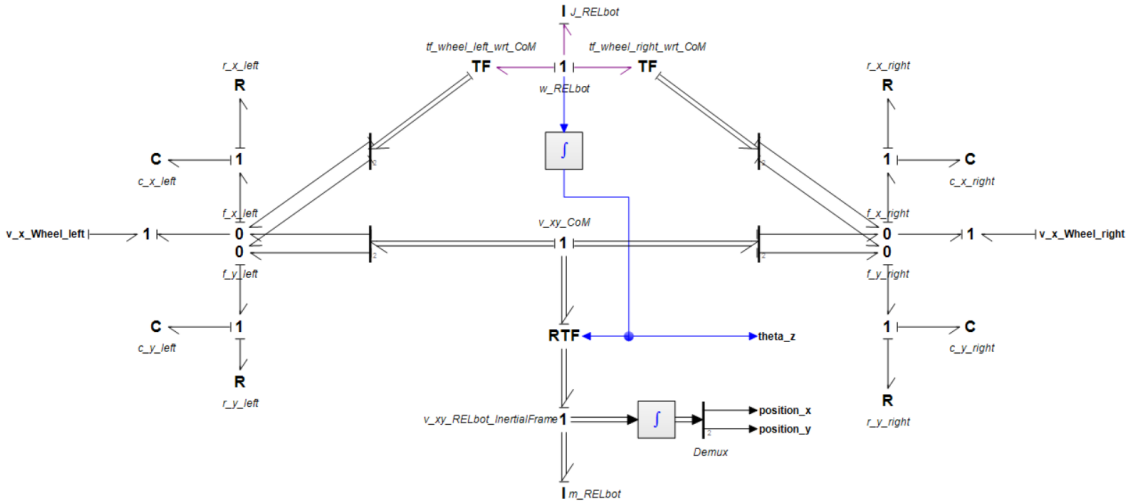

Figure D.5: Rigid body bondgraph

E 20-Sim controller test

Goal and setup

The goal of this test is to find the performance of the PID controller in 20-Sim. A position reference profile is given to the controller. The controller generates control values so that the position of the plant corresponds to the position reference. The difference between the position setpoint and the measured position is a measure of controller performance.

Results

Figure E.1 shows the position setpoint and the plant position in the top plot and the difference between the two in the bottom plot.

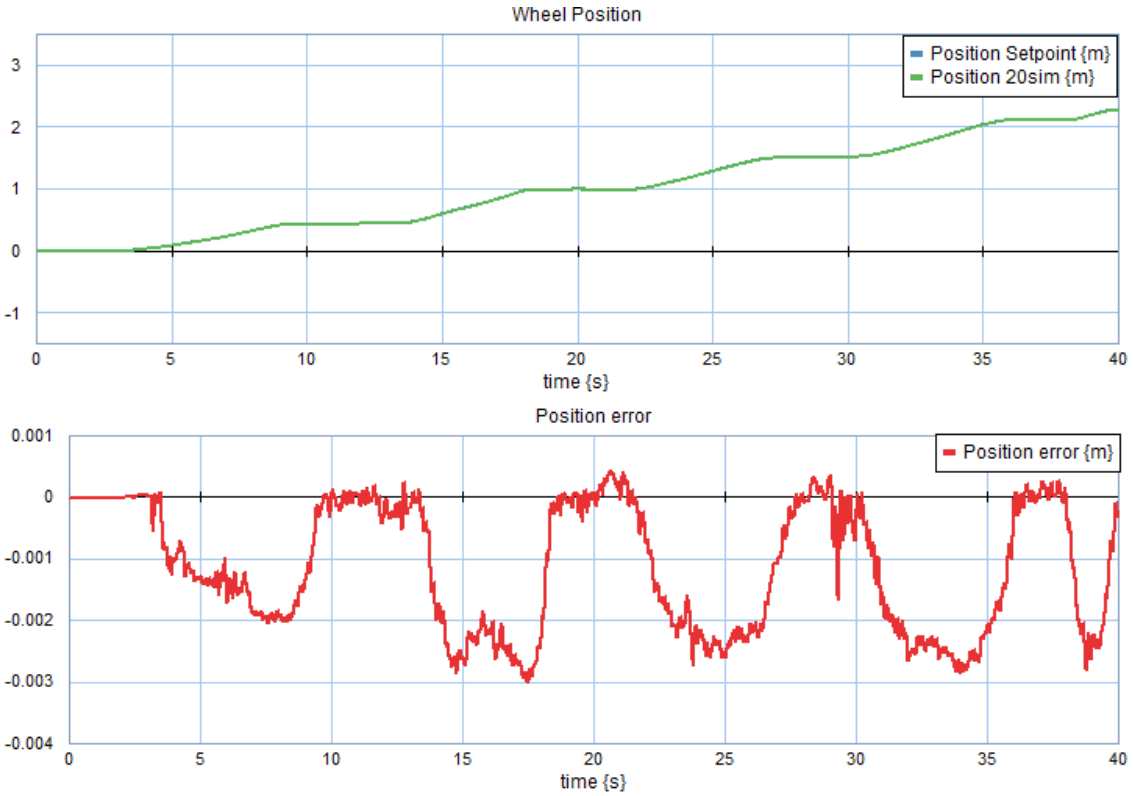

Figure E.1: Closed-loop controller test in 20-Sim

Interpretation

The plot shows that the controller works well since the position error remains below 3 mm and the controller has no steady state error. Therefore the controller is good enough.

F Test setups

Open-loop HiL vs 20-Sim test

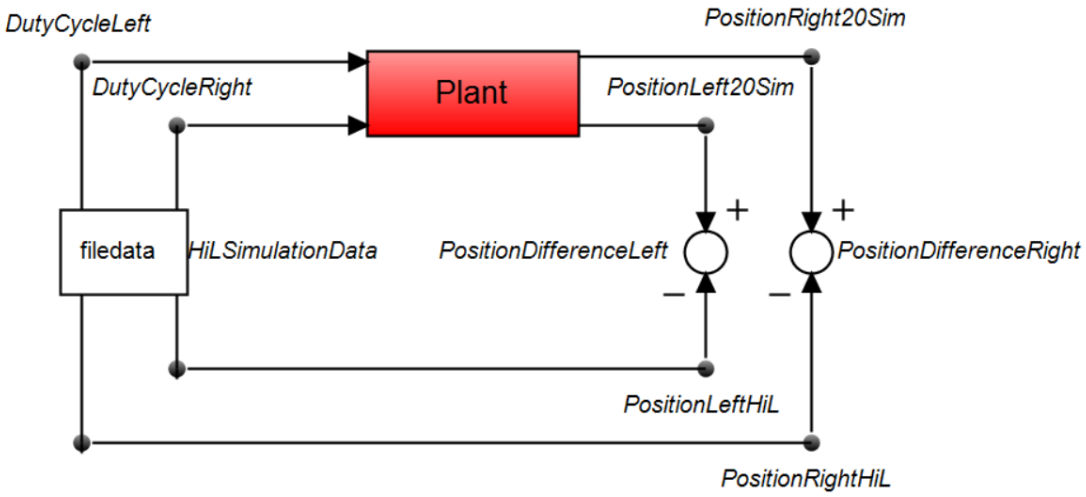

Figure F.1: 20-Sim model for the open-loop HiL vs 20-Sim test

Open-loop HiL vs 20-Sim test with delay compensation

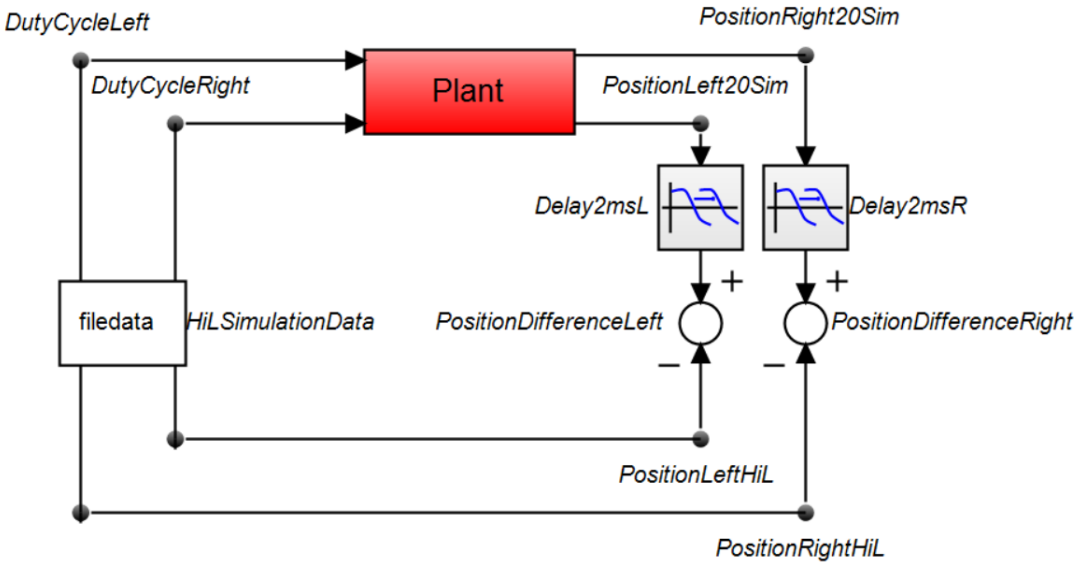

Figure F.2: 20-Sim model for the open-loop HiL vs 20-Sim test with delay compensation

RELbot vs HiL simulation open-loop comparison

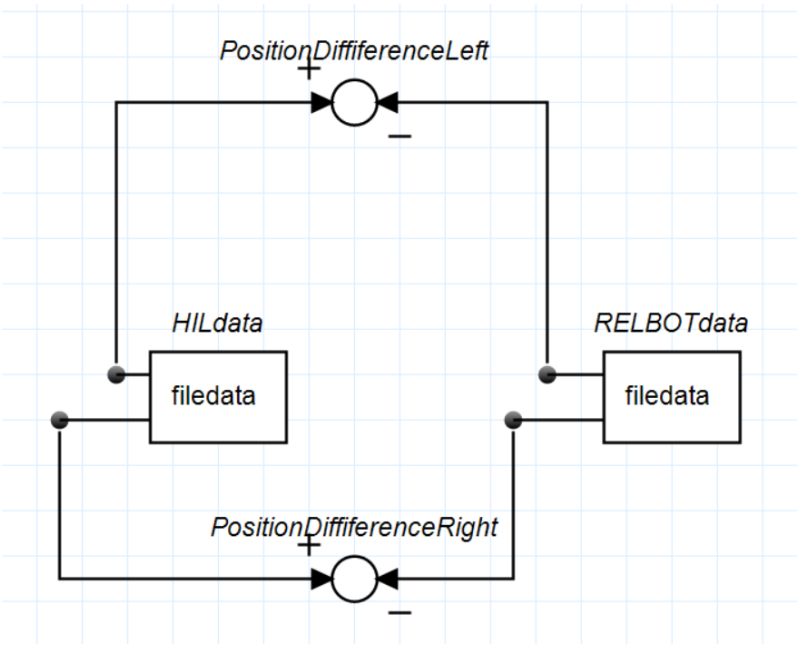

Figure F.3: 20-Sim model for the RELbot vs HiL simulation open-loop test

RELbot vs HiL simulation closed-loop comparison

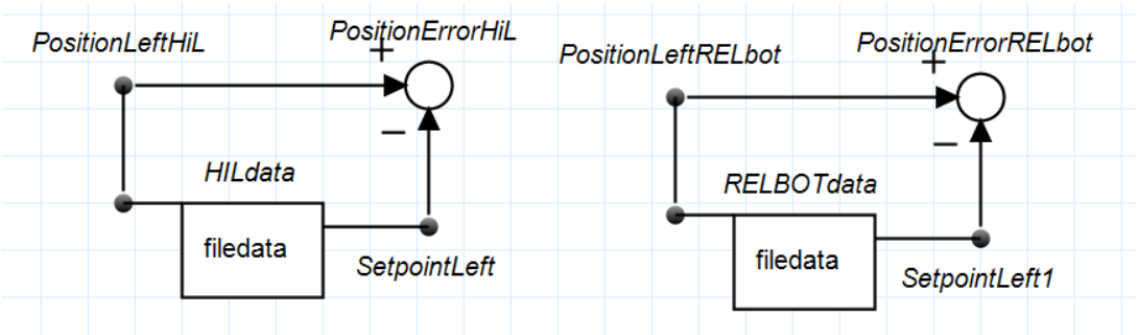

Figure F.4: 20-Sim model for the RELbot vs HiL simulation closed-loop test

G Digital Twin input output conversion code

FPGA PWM to dutycycle conversion

```
u[0] = ((double)sample_data.channel2)/PWM_RESOLUTION; //Left
u[1] = ((double)sample_data.channel1)/PWM_RESOLUTION; //Right
```

Output wheel position to FPGA Encoder position

```
//meters to wheel_radians to motor_radians to encoder_counts
encoder_output_left = y[0]/wheel_radius/gear_ratio
   *encoder_counts_per_rotation;
encoder_output_right = y[1]/wheel_radius/gear_ratio
   *encoder_counts_per_rotation;

actuate_data.enc1 = static_cast<int16_t>(encoder_output_right);
actuate_data.enc2 = static_cast<int16_t>(encoder_output_left);
```

Absolute encoder position to relative encoder position

The following code makes sure the encoder position is between -8192 and +8191. Depending on the encoder resolution.

```
InvIcoIo::WriteValue InvIcoIo::calc_value(int enc_val, bool
   pin_val)
{
    WriteValue write_value;
    write_value.fill = 0;

    while(enc_val >= encoder_resolution/2){
        enc_val = enc_val - encoder_resolution;
    }
    while(enc_val < -encoder_resolution/2){
        enc_val = enc_val + encoder_resolution;
    }
    write_value.enc_val = enc_val;

    write_value.pin_val = static_cast<unsigned int>(pin_val);
    return write_value;
}
```

References

Březina, Tomáš and Ryszard Jabłoński, eds. (2018). *Mechatronics 2017: Recent Technological and Scientific Advances*. Vol. 644. Advances in Intelligent Systems and Computing. Cham: Springer International Publishing. ISBN: 978-3-319-65959-6 978-3-319-65960-2.

Broenink, Jan F. (2020). 'Bond Graphs: A Unifying Framework for Modelling of Physical Systems'. In: *Foundations of Multi-Paradigm Modelling for Cyber-Physical Systems*. Ed. by Paulo Carreira, Vasco Amaral and Hans Vangheluwe. Cham: Springer International Publishing, pp. 15–44. ISBN: 978-3-030-43946-0.

Broenink, Jan F. and Yunyun Ni (2012). 'Model-driven robot-software design using integrated models and co-simulation'. In: pp. 339–344. DOI: 10.1109/SAMOS.2012.6404197.

Dai, Xunhua, Chenxu Ke, Quan Quan and Kai-Yuan Cai (Apr. 2021). 'Simulation Credibility Assessment Methodology With FPGA-based Hardware-in-the-Loop Platform'. In: *IEEE Transactions on Industrial Electronics* 68.4, pp. 3282–3291. ISSN: 0278-0046, 1557-9948.

Delgado, Raimarius, Bum-Jae You and Byoung Wook Choi (May 2019). 'Real-time control architecture based on Xenomai using ROS packages for a service robot'. In: *Journal of Systems and Software* 151, pp. 8–19. ISSN: 01641212.

Donglin Xiu, X.S. Hu, M.D. Lemmon and Qiang Ling (2003). 'Firm real-time system scheduling based on a novel QoS constraint'. In: *Proceedings. 2003 International Symposium on System-on-Chip (IEEE Cat. No.03EX748)*. Cancun, Mexico: IEEE Comput. Soc, pp. 386–395. ISBN: 978-0-7695-2044-5.

Grieves, Michael (2016). *Origins of the Digital Twin Concept*. DOI: 10.13140/RG.2.2.26367.61609.

Groothuis, M. (2004). *Distributed HIL simulation for Boderc*. MSc Thesis. University of Twente.

Hamdaoui, M. and P. Ramanathan (1995). 'A dynamic priority assignment technique for streams with (m, k)-firm deadlines'. In: *IEEE Transactions on Computers* 44.12, pp. 1443–1451. DOI: 10.1109/12.477249.

Hooglander, Frank (Dec. 2023). *Emulation of quadrature and H-bridge signals on an FPGA to enable HiL simulation*. Individual assignment. University of Twente.

Isermann, R, J Schaffnit and S Sinsel (1999). 'Hardware-in-the-loop simulation for the design and testing of engine-control systems'. In: *Control Engineering Practice*. ISSN: 0967-0661.

Pop, Adrian Ioan, Nicolae Pop, Radu Ţarcă, Claudiu Lung and Sebastian Sabou (June 2023). 'Wheeled Mobile Robot H.I.L. Interface: Quadrature Encoders Emulation With A Low-Cost Dual-Core Microcontroller'. In: *2023 17th International Conference on Engineering of Modern Electric Systems (EMES)*. Oradea, Romania: IEEE, pp. 1–4. ISBN: 9798350310634.

Raoudi, Ilyas (2024). *ROS2 – Xenomai4 Real-Time Framework on Raspberry Pi*. MSc Thesis. University of Twente.

Widiarta, Gede Haris, Muhammad Zakiyullah Romdlony, Muhammad Ridho Rosa and Bambang Riyanto Trilaksono (Nov. 2021). 'Control Lyapunov — Barrier Function Implementation for Mobile Robot Model with Hardware in the Loop'. In: *2021 International Conference on Mechatronics, Robotics and Systems Engineering (MoRSE)*. Bali, Indonesia: IEEE, pp. 1–6. ISBN: 978-1-66540-844-8.

Wielink, Lucas (2024). *Modelling and simulation of the RELbot*. BSc Thesis. University of Twente.

Yeh, Loh Chow and Hermawan Nugroho (June 2021). 'Design of Hardware-in-the-loop Simulation Approach for Slip-Compensated Odometry Tracked Mobile Robot Platform'. In: *2021 8th International Conference on Computer and Communication Engineering (ICCCE)*. Kuala Lumpur, Malaysia: IEEE, pp. 355–360. ISBN: 978-1-72811-065-3.